

Produktionsinformatik

Eric MSP Veith e.veith@hs-osnabrueck.de

March 6, 2019

Outline

In Medias Res: Metaprognose

Ein kurzer Blick auf die Geschichte und der Status Quo

Datenhaltung

Schnellkurs: Python for Data Scientists

Datenexploration und -vorverarbeitung

Künstliche Neuronale Netze: Basiswissen

Muster- und Objekterkennung

Training künstlicher neuronaler Netze

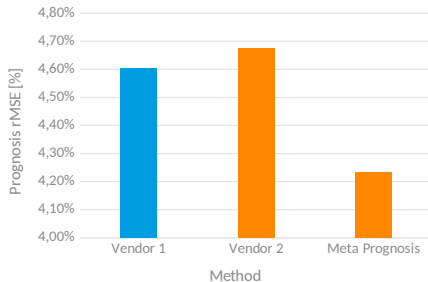
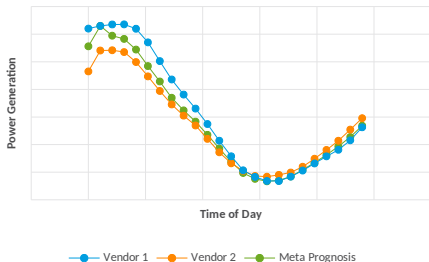
Zeitreihenvorhersage mit rekurrenten neuronalen Netzen

Literaturempfehlungen

In Medias Res: Metaprognose

Metaprognose

Spezifikation und Ziel



- Intelligentes „Verschneiden“ zweier Prognosen
- Fehler spezifisch für Portfolio: „Finetuning“

Jupyter-Notebook

jupyter pv_metaforecast Last Checkpoint: Last Wednesday at 2:59 PM (autosaved)  Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Zusammenfassung

Das Ergebnis der ersten Übung?

- Jedes Deep-Learning-Modell ist **nur so gut wie die ihm zugrundeliegenden Daten**.
- Deep Learning $\neq$ Magie
- Struktur vorhanden/vermutet, Quantifizierung gesucht?
Ein Fall für **Machine Learning** (lineare Regression, KNN, ...)
- Quantifizierung komplex, viele Sonderfälle, komplexe Struktur?
Ein Fall für **Deep Learning**

Ein kurzer Blick auf die Geschichte und der Status Quo



„Künstliche Intelligenz ist die Wissenschaft davon,
Maschinen Dinge tun zu lassen,
deren Ausführung durch einen Menschen
Intelligenz erfordern würde.“

— Marvin Minsky, 1966

KI, ML und DL — „Was zum...?“

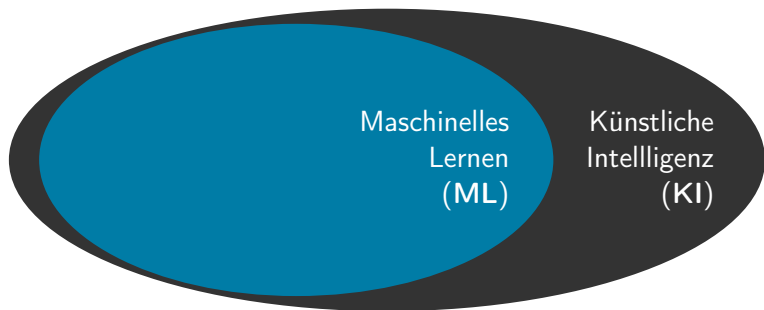
Einordnung künstliche Intelligenz, maschinelles Lernen und Deep Learning



Künstliche
Intelligenz
(KI)

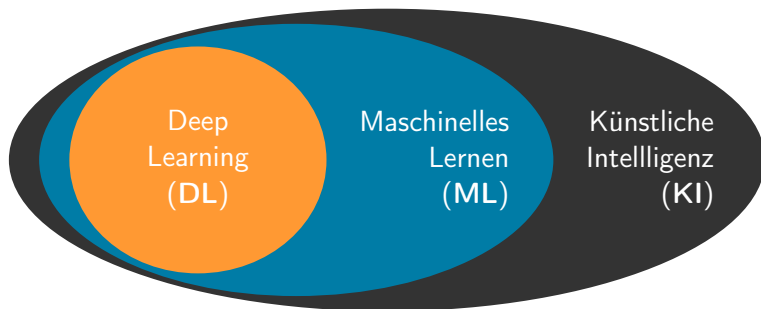
KI, ML und DL — „Was zum...?“

Einordnung künstliche Intelligenz, maschinelles Lernen und Deep Learning



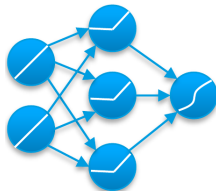
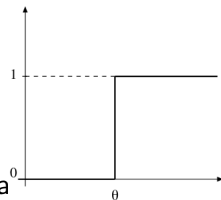
KI, ML und DL — „Was zum...?“

Einordnung künstliche Intelligenz, maschinelles Lernen und Deep Learning



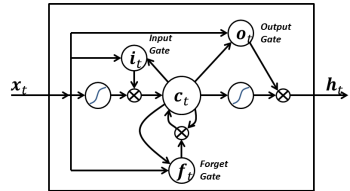
Die Anfänge

- Entstehung künstlicher neuronaler Netze
 - 1943 **Threshold Logic**: McCulloch & Pitts
 - 1949 **Hebbsche Lernregel**: Hebb
- Erste Blüte
 - 1956 **Dartmouth Conference**: McCarthy et. al.
 - 1958 **Perceptron**: Rosenblatt
 - 1969 **Perceptron-Analyse**: Minsky & Papert
- Erster „KI-Winter“
- Wiederaufbau & Renaissance
 - 1982 **Backpropagation of Error**: Werbos
 - 1985 Rumelhart
- Zweiter „KI-Winter“



Vor dem *Deep Learning*

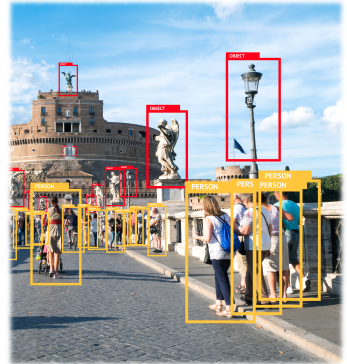
- Vielfältige Einsatzmöglichkeiten, z. B.
 - Handschriftenerkennung
 - Bild-/Objekterkennung
 - Mustererkennung
 - Natural Language Processing
- Kein durchschlagender Erfolg
 - Mangelnde Rechenleistung:
kleine Mustermengen zum Training
 - Hand-Vorauswahl relevanter Muster
 - Stark spezialisiert
 - Probleme der Lernalgorithmen mit Extremfällen
- Grundlage für besseres Lernen: Tauwetter
1997 **Long Short-term Memory**: Hochreiter & Schmidhuber
- Immer noch Mangel an Rechenleistung für notwendige
Mustermengen



BiObserver — CC-BY-SA 4.0

Tiefes Lernen

- Neuer Enthusiasmus: Durchbruch dank GPGPU
 - 2007 **CUDA**: nVidia
 - 2013 **LeNet 5**: LeCunn et al.
 - 2016 **AlphaGo**: Hassabis et al.
- „Ab wann ist maschinelles Lernen *tief*?“
 - Komplexe künstliche neuronale Netze (viele Schichten ~ viele Features)
 - Keine harte Grenze zwischen *Shallow* und *Deep Learning*



Arten des Lernens

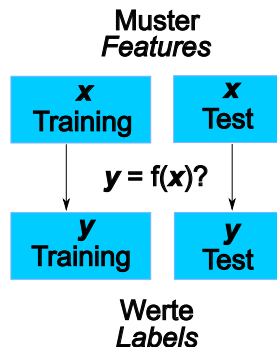
Aufgaben für ein künstliches neuronales Netz

- **Modellbasiertes Lernen: ANN**
entwickelt Modell des Problems (vs. Instanzbasiertes Lernen)
- **Supervised Learning**
 - Klassifikation
 - Regression
- **Unsupervised Learning**
 - Clustering
- **Reinforcement Learning**

Arten des Lernens

Aufgaben für ein künstliches neuronales Netz

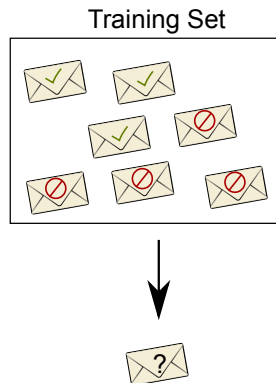
- **Modellbasiertes Lernen:** ANN entwickelt Modell des Problems (vs. Instanzbasiertes Lernen)
- **Supervised Learning**
 - Klassifikation
 - Regression
- **Unsupervised Learning**
 - Clustering
- **Reinforcement Learning**



Arten des Lernens

Aufgaben für ein künstliches neuronales Netz

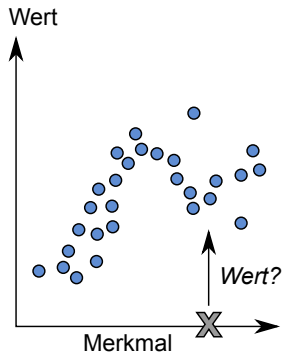
- **Modellbasiertes Lernen: ANN**
entwickelt Modell des Problems (vs. Instanzbasiertes Lernen)
- **Supervised Learning**
 - Klassifikation
 - Regression
- **Unsupervised Learning**
 - Clustering
- **Reinforcement Learning**



Arten des Lernens

Aufgaben für ein künstliches neuronales Netz

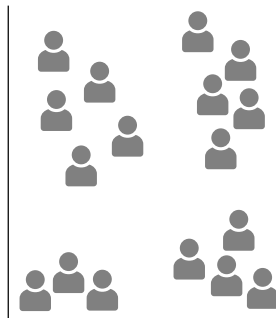
- **Modellbasiertes Lernen:** ANN entwickelt Modell des Problems (vs. Instanzbasiertes Lernen)
- **Supervised Learning**
 - Klassifikation
 - Regression
- **Unsupervised Learning**
 - Clustering
- **Reinforcement Learning**



Arten des Lernens

Aufgaben für ein künstliches neuronales Netz

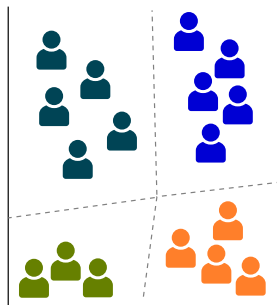
- **Modellbasiertes Lernen: ANN**
entwickelt Modell des Problems (vs. Instanzbasiertes Lernen)
- **Supervised Learning**
 - Klassifikation
 - Regression
- **Unsupervised Learning**
 - Clustering
- **Reinforcement Learning**



Arten des Lernens

Aufgaben für ein künstliches neuronales Netz

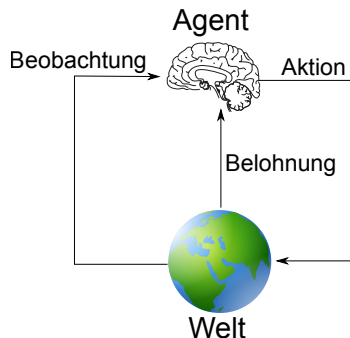
- **Modellbasiertes Lernen:** ANN entwickelt Modell des Problems (vs. Instanzbasiertes Lernen)
- **Supervised Learning**
 - Klassifikation
 - Regression
- **Unsupervised Learning**
 - Clustering
- **Reinforcement Learning**



Arten des Lernens

Aufgaben für ein künstliches neuronales Netz

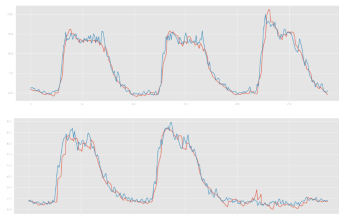
- **Modellbasiertes Lernen:** ANN entwickelt Modell des Problems (vs. Instanzbasiertes Lernen)
- **Supervised Learning**
 - Klassifikation
 - Regression
- **Unsupervised Learning**
 - Clustering
- **Reinforcement Learning**



Arten des Lernens

Aufgaben für ein künstliches neuronales Netz

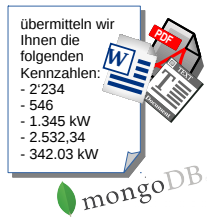
- **Modellbasiertes Lernen: ANN**
entwickelt Modell des Problems (vs. Instanzbasiertes Lernen)
 - **Supervised Learning**
 - Klassifikation
 - Regression
 - **Unsupervised Learning**
 - Clustering
 - **Reinforcement Learning**
- ...vor allem **Supervised Learning**
interessant



Datenhaltung

Unstrukturierte Daten

- Kein Datenmodell
- Keine Metainformationen
- Hoher Aufwand beim Einlesen (Parsen); zeit- und fehleranfällig



```
import re
```

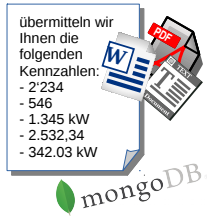
```
pat = re.compile("^- ([-0-9.,']+)")  
with open('foo') as fp:  
    for line in fp:  
        m = pat.search(line)  
        kw = int(m.group(0)) if m
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>  
ValueError: invalid literal for int()  
with base 10: "2'234"
```

Unstrukturierte Daten

- Kein Datenmodell
- Keine Metainformationen
- Hoher Aufwand beim Einlesen (Parsen); zeit- und fehleranfällig



Excel

- Annähernd strukturiert
- Metainformationen
- Achtung: i18n:
 - Zahlenformate
 - Namen von Tabellen („Tabelle1“)
- Proprietäres Datenformat:
:- (

```
import openpyxl
from openpyxl import Workbook
wb =
    ↪ openpyxl.load_workbook(data_file,
    ↪ data_only=True)
ws = wb['Tabelle1']
ws['A1'] = datetime.datetime(2018,
    ↪ 3, 18)
ws['A1'].number_format
# => 'yyyy-mm-dd h:mm:ss'
wb.guess_types = True
```

- Bei weitem nicht auf Komma und Zeilenumbruch beschränkt
- Offenes Format:
 - Text
 - Trenner für Datensätze, Datenfelder und Feldbegrenzer
 - Keine Standardisierung
- Mit das am häufigsten vorkommende Datenformat
- Problem: Interpretation (kein Datenmodell, keine Metainformationen)

4; 54,2 kW; 8:09; ok

...

15 5 00:15

....

"3", "4,56", "14:35", "\\\"Abschluss\\\""

HDF5

Hierarchical Data Format

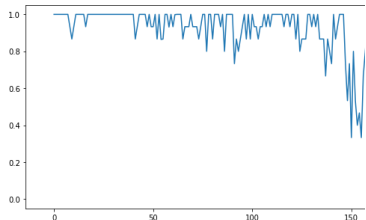
- Vor allem in Forschung präsent
- Effizient auch bei großen Datenmengen
- Hierarchisches Dateiformat
 - *Dataset*: Mehrdimensionale Matrix eines homogenen Datentyps
 - *Group*: Container für *Datasets* und andere Gruppen
- Offener Standard (interoperabel)
- Vergleichsweise geringe Verbreitung in der Geschäftswelt

```
In [21]: %matplotlib inline
import h5py
import numpy as np
import matplotlib.pyplot as plt

In [20]: h5 = h5py.File('wind_braunschweig.hdf5', 'r')
for k, v in h5['feedin'].attrs.items():
    print("%s = %s" % (k, v))

start = 1293836400
resolution = 900

In [24]: fig, ax = plt.subplots(figsize=(15, 5))
_ = ax.plot(h5['feedin'][0:4*72])
```



Was ist nun Big Data...?

Wesentliche Elemente

- Volume Große, stetig wachsende Datenmengen im hohen Terabyte-/Petabyte-Bereich
- Velocity Daten werden mit hoher Frequenz generiert, Behandlung nahe Echtzeit nötig
- Variety Keine uniformen Datenformate, sondern sogar teils unstrukturierte Daten
- Veracity Daten sind auch fehlerbehaftet/nicht vorverarbeitet und aufbereitet
- Value Daten allein sind noch kein Mehrgewinn; Big Data ist eine „Zubringertechnologie“

PostgreSQL

The Elephant in the Room

- Wohlbekannt (geringe Schulungskosten!)
- alle Vorteile einer SQL-Datenbank
- Auch Dokumente via JSON
 - `'{"a":1,"b":2}'::json->'b'`
- LISTEN/NOTIFY für asynchrone Arbeit
- Schemata: Daten in gutem Format für Deep Learning



Asynchrone Verarbeitung mit PostgreSQL

LISTEN/NOTIFY

- `LISTEN channel;`
lauscht auf Kanal (erzeugt, falls nötig)
- `NOTIFY channel payload;`
benachrichtigt alle Zuhörer (payload optional)
- Kombination mit asynchroner Behandlung sinnvoll, z. B. `select(·)`

Asynchrone Verarbeitung mit PostgreSQL

LISTEN/NOTIFY

```
import select; import psycopg2; import psycopg2.extensions

conn = psycopg2.connect(DSN)
conn.set_isolation_level(
    psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
curs = conn.cursor()
curs.execute("LISTEN somechannel;")

while True:
    if select.select([conn], [], [], TIMEOUT) != ([], [], []):
        conn.poll()
        while conn.notifies:
            notify = conn.notifies.pop(0)
            print("Got NOTIFY: ", notify.pid,
                  notify.channel,
                  notify.payload)
```

Nachteile von PostgreSQL

“Not quite big data”

- Schemata: Vorverarbeitung nötig
- Abfragegeschwindigkeit abhängig von Tabellengröße

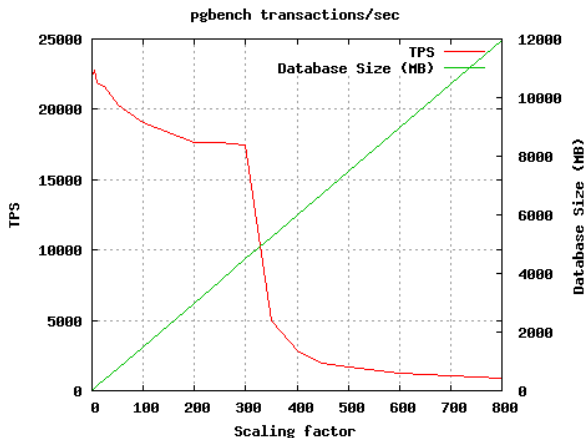


Table Partitioning

Große Datenmengen effizient Speichern

- Max. Größe einer Tabelle: 32 TB
- Je größer die Tabelle, desto kleiner die Abfragegeschwindigkeit
- Lösung: **Table Partitioning**

```
CREATE TABLE consumption (  
    id SERIAL,  
    sensor_id INT NOT NULL,  
    log_ts TIMESTAMP WITH TIME ZONE NOT NULL,  
    value REAL NOT NULL)  
PRIMARY KEY (id)  
PARTITION BY RANGE (log_ts);
```

```
CREATE TABLE consumption_y2006m02 PARTITION OF consumption  
FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');
```

Table Partitioning

Automatisches Partitionieren über Funktion

```
CREATE OR REPLACE FUNCTION create_partition_and_insert()
RETURNS trigger AS
$BODY$
DECLARE partition_date TEXT; partition TEXT;
BEGIN
    partition_date := to_char(NEW.log_ts, 'YYYY_MM_DD');
    partition := TG_RELNAME || '_' || partition_date;
    IF NOT EXISTS(SELECT relname FROM pg_class WHERE relname=partition)
        EXECUTE 'CREATE TABLE ' || partition
            || ' INHERITS (' || TG_RELNAME || ');';
    END IF;
    EXECUTE 'INSERT INTO ' || partition
        || ' SELECT(' || TG_RELNAME || ' ' || quote_literal(NEW)
        || ').* RETURNING id;';
    RETURN NULL;
END;
$BODY$
LANGUAGE plpgsql VOLATILE COST 100;
```

Table Partitioning

Automatisches Partitionieren über Funktion

```
CREATE TRIGGER measurements_partition_insert_trigger
BEFORE INSERT ON measurements
FOR EACH ROW EXECUTE PROCEDURE
    create_partition_and_insert();
```

- Partitionen normalerweise manuell angelegt
- Alternative: Trigger (wie oben), verlangsamt aber Einfügevorgang
- **Default Partition** möglich

```
CREATE TABLE measurement_default
PARTITION OF measurement DEFAULT;
```


Table Partitioning

Vorteile

- Speichermedium je nach Aktualität der Partition (z. B. HDD vs. SSD)

- Weitehin Indices, Trigger, etc. möglich

```
CREATE INDEX ON consumption (log_ts);
```

```
CREATE INDEX ON consumption (sensor_id);
```

- UPDATE verschiebt Datensätze zwischen Partitionen

- Auch Subpartitionen möglich

```
CREATE TABLE consumption_y2006m02
```

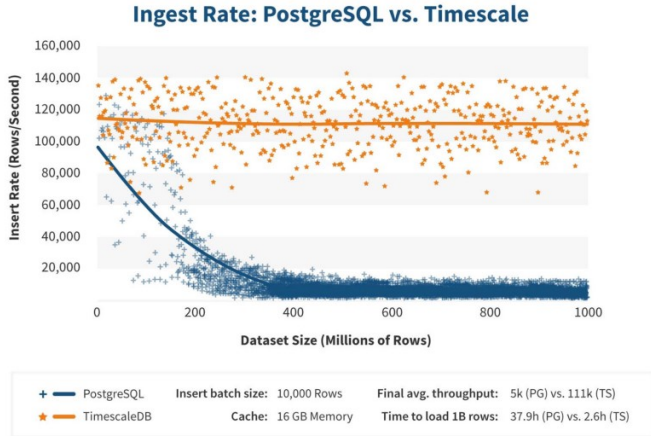
```
PARTITION OF consumption
```

```
FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');
```

```
PARTITION BY RANGE (sensor_id);
```

Table Partitioning

Nachteile



© 2019 Timescale, Inc.

TimescaleDB

PostgreSQL mit Zeitreihendaten

- Viele Daten im Deep-Learning-Kontext sind **Zeitreihendaten**
- **Zeitzentriert**: Immer mit Zeitstempel
- **Append-only**: Daten werden angehängt, selten verändert, nie gelöscht
- **Recency**: Neue Daten beziehen sich fast ausschließlich auf aktuelle Zeitintervalle; kein Auffüllen

Timestamp	Power	Wind Speed	Wind Dir.
2014-10-23T21:10:00+01:00	223.12	348.65	4.98
2014-10-23T21:20:00+01:00	149.22	349.00	4.42
2014-10-23T21:30:00+01:00	181.65	349.10	4.74

Datenorganisation von TimescaleDB

Eine Erweiterung von PostgreSQL

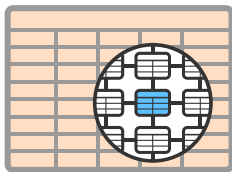
- Benutzung sehr einfach:

```
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;  
SELECT create_hypertable('measurements', 'log_ts');
```

- Zeitreihentabelle (**Hypertable**) besteht aus mehreren disjunkten **Chunks**
- Chunks: rechtslastige B-Bäume
- Chunks sollten in den Arbeitsspeicher passen (ca. 25% RAM für die neusten Chunks); Zeitintervall entsprechend wählen



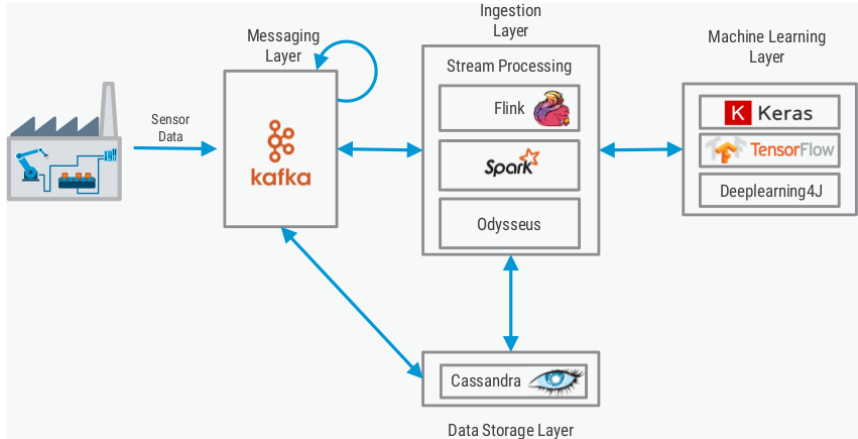
Hypertable



Chunk

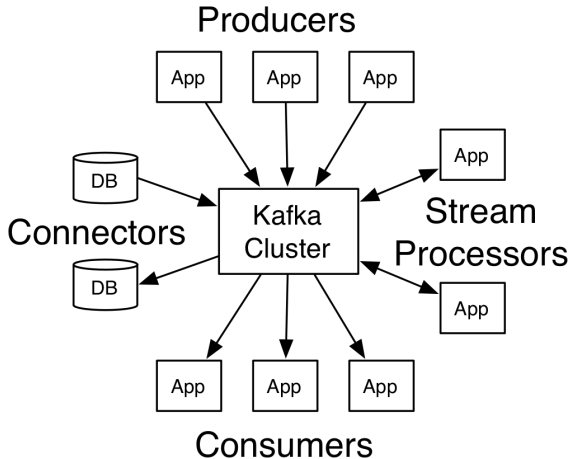
Ein Big Data Stack

The Big Picture



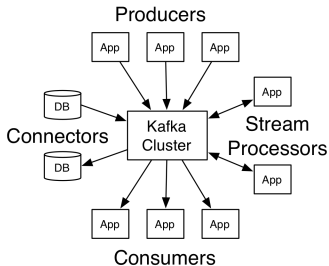
Apache Kafka

Distributed Streaming Platform



Apache Kafka

In a Nutshell



- **Distributed Streaming Platform:**
Effizienter Nachrichtenverteiler
- Nachrichtenverteilung über **Producer-Consumer-Muster**
- Einfaches Nachrichtenformat:
 $m = (key, value)$
- *Topics* zur Organisation:
 $\{m_1, m_2, \dots, m_n\}$
- Elegante Enkopplung verschiedener Komponenten im *Big Data Stack*
- Unterstützt auch
Microservices-Architekturen

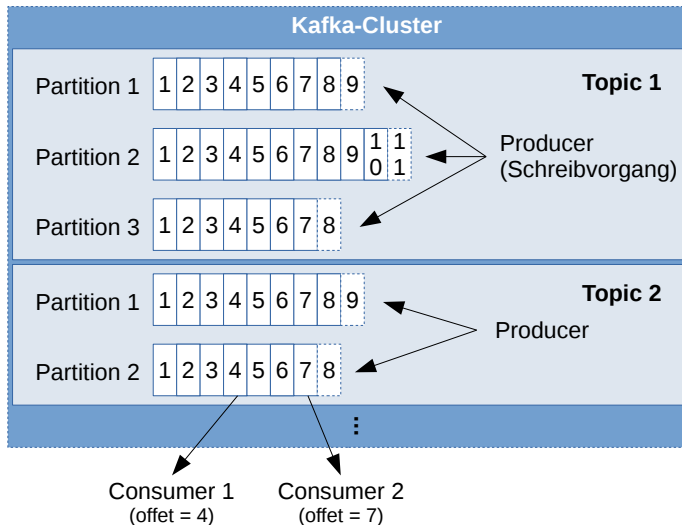
Apache Kafka: Architektur

Producer, Consumer & Partitionen

- **Topics:** geordnet Strom von **Records**
- Records werden von einem **Producer** geliefert
- Ein Topic hat beliebig viele **Consumer**
- Ein Topic besteht aus beliebig vielen **Partitionen**
- **Partition:** Geordnete, unveränderliche Sequenz von Records (**append only**)
- Eine Partition muss auf einem Server Platz finden; ansonsten für Clustering genutzt
- Records verbleiben bis zum Ablauf der **Retention Period** in einer Partition
- Einzelne Consumer haben ein individuelles **Offset**
- Quintessenz: Kafka ist eine Kombination aus **Nachrichtendienst**, **Nachrichtenspeicher** und **Datenstrom**

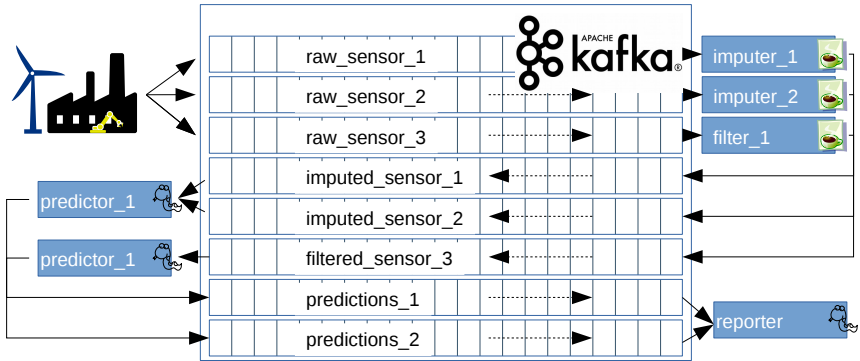
Apache Kafka: Architektur

Producer, Consumer & Partitionen eines Clusters



Einsatz von Kafka

Messaging Services als essenzieller Teil einer Big Data-Plattform



- Kafka erlaubt Segmentierung, damit Aufteilung auf verschiedene Geräte und ggfs. verschiedene Programmiersprachen
- Begünstigt **Microservices-Architekturen**

Producer für Apache Kafka

Minimal Working Example in Python

```
import asyncio; from aiokafka import AIOKafkaProducer
event_loop = asyncio.get_event_loop()

async def say_hello():
    producer = AIOKafkaProducer(loop=event_loop,
                                bootstrap_servers='br1,br2',
                                acks='all')    # 0, 1, all
    await producer.start()    # Wait for Future
    try:
        await producer.send_and_wait("my_topic",
                                     b"Hello, World!")
    finally:
        await producer.stop()

event_loop.run_until_complete(say_hello())
```

Consumer für Kafka

Minimal Working Example in Python für asynchrone Nachrichtenbehandlung

```
consumer = aiokafka.AIOKafkaConsumer("my_topic",
    loop=event_loop,
    bootstrap_servers='localhost:9092',
    group_id="my_group",
    enable_auto_commit=True,
    auto_commit_interval_ms=1000)
await consumer.start()
try:
    async for msg in consumer:
        print("{}: {:d}: {:d}: key={} value={} timestamp_ms={}" \
            .format(msg.topic, msg.partition, msg.offset,
                msg.key, msg.value, msg.timestamp))
finally:
    await consumer.stop()
```

■ Veracity & Value:

1. Daten vorverarbeiten: Ausreißer säubern, fehlende Werte imputieren, ggfs. normieren/zentrieren, ...
2. Verwertung zuführen: ML-Algorithmen benötigen *immer* vorverarbeitete Daten (bspw. äquidistante Zeitreihen)

■ Spark arbeitet auf **Mini Batches**

■ Eigene Variante von SQL, **Spark SQL**

■ Klassen für maschinelles Lernen (**Spark ML**), z. B.

- ☐ Scaler
- ☐ Imputer
- ☐ PCA
- ☐ ...

„Hello World“ in Apache Spark

Minimal Working Example für Data Frames

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split

spark = SparkSession.builder \
    .appName("StructuredNetworkWordCount") \
    .getOrCreate()

lines = spark.readStream.format("socket") \
    .option("host", "localhost").option("port", 9999) \
    .load()

words = lines.select(explode(split(lines.value, " ")) \
    .alias("word"))

wordCounts = words.groupBy("word") \
    .count()
```

Imputer

Parameter & Verwendung

- In Sensor-Datenströmen können Werte fehlen (Sensorausfall, etc.)
- Existierende Werte als Stützwerte
- Imputationsstrategie:
 - Durchschnitt (Voreinstellung — Achtung, begünstigt Ausreißer!)
 - Median
- Ablauf: `fit(.) -> model`, dann `model.transform(df)`

Vorverarbeitung mit Apache Spark

```
# ...  
from pyspark.ml.feature import Imputer  
  
df = spark.readStream.format("kafka") \  
    .option("kafka.bootstrap.servers",  
            "host1:port1,host2:port2") \  
    .option("subscribe", "raw_sensor_1") \  
    .load()  
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")  
  
imputer = Imputer(inputCols=["value"], outputCols=["value_out"])  
model = imputer.fit(df)  
model.transform(df).show()
```


Cassandra & Kafka

Details

Apache Cassandra: NoSQL-Datenbank

- Strukturiert
- Spaltenorientiert
- Schlüssel-Werte-Paare

Apache Kafka: **Stream Processor**

- Effiziente Nachrichtenverteilung:
Producer-Consumer-Muster
- Nachricht: $m = (key, value)$
- *Topics* zur Organisation:
 $\{m_1, m_2, \dots, m_n\}$

Cassandra & Kafka

Ein Code-Beispiel

```
cluster = cassandra.cluster.Cluster()
session = cluster.connect('test')
kafka = kafka.client.KafkaClient("localhost:9092")

consumer = SimpleConsumer(kafka, "test-group",
    ↪ "cassandratest")
model = keras.model.Sequential() # ; add(...);
    ↪ add(...); fit(...)
insert_stmt = session.prepare("insert into simple (id,
    ↪ message) values (?,?)");

for message in consumer:
    y = model.predict(message.message.value)
    session.execute(insert_stmt,
        ↪ [str(datetime.datetime.now()), y])
```

Hadoop

- Framework für **verteilt arbeitende Software**
- Dateisystem: **HDFS**
- Ausführungsschema: **gerichtet, azyklischer Graph**
- Interessant ab Datenmengen im **Tera-/Petabyte-Bereich**

```
% bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar \  
-file /home/user/mapper.py      -mapper /home/user/mapper.py \  
-file /home/user/reducer.py     -reducer /home/user/reducer.py \  
-input /user/user/indata/*      -output /home/user/output
```

Zusammenfassung

Präferenzen bei Datenformaten?

- **Pragmatismus:** Nutzen Sie, was funktioniert und was die Datenquelle liefern kann!
- **Kommunikation:** Reden Sie mit dem Datenlieferanten!
- **Dokumentation:** Schnittstellen sind nur so gut wie ihre Beschreibung.

Wenn Auswahl besteht, dann...

- Explorativ:
 1. **Kleine Datenmengen:** HDF5, CSV
 2. **Große, homogene/vorverarbeitete Daten:** HDF5
- Produktiv:
 1. HDF5
 2. **TB–PB Daten, aktive Datenquelle:** Kafka & Cassandra
 3. **PB Daten, gerichtete Verarbeitung:** Hadoop
 4. CSV

In den meisten Fällen: CSV. :-|

Machine Learning Project Checklist

- Das Problem eingrenzen, Zusammenhänge verstehen
- Daten beschaffen (an die Daten-Pipeline dabei denken)
- Daten explorieren
- Datenvorverarbeitung, um sie für ML-Algorithmen besser nutzbar zu machen
- Verschiedene Modelle testen
- Fine-tuning der besten Modelle, Kombination zu einer Lösung
- Präsentation, Überführung in den Produktiveinsatz, Überwachung & Wartung

Schnellkurs: Python for Data Scientists

Python

Die *Lingua Franca* des *Deep Learning*

- Entwickelt 1991 von Guido van Rossum
- Leicht zugänglich;
“*There is one—and preferable only one—way to do things.*”
- Versatile Programmiersprache, u. a. für UNIX-Systemwerkzeuge, Webanwendungen und wissenschaftliche Analysewerkzeuge
- Dynamische Typisierung (*Duck Typing*)
- Viele hochoptimierte (C/C++-Kern, Python-API)
(z. B. NumPy, SciPy, TensorFlow, ...)

Python Byte-for-Byte

Syntax, Primitive & Verzweigungen

- Einfache Syntax, basierend auf **Einrückung** und Zeilenumbrüchen
- Skalare (primitive Datentypen)

$a = 42, 13, 37, \pi, \sqrt{2}, \sqrt{-1}, \dots$

```
>>> a = 42.0
42.0
>>> type(a)
<class 'float'>
>>> a += 0.3
>>> a
42.3
```


Python Byte-for-Byte

Syntax, Primitive & Verzweigungen

- Einfache Syntax, basierend auf **Einrückung** und Zeilenumbrüchen
- Skalare (primitive Datentypen)

$a = 42, 13, 37, \pi, \sqrt{2}, \sqrt{-1}, \dots$

```
>>> a = 42.0
42.0
>>> type(a)
<class 'float'>
>>> a += 0.3
>>> a
42.3
```

- Verzweigungen

```
person = "Viking"
if person == "Viking":
    print("Spam, spam, spam, lovely spam!")
```

Listen in Python

Definition, Zugriff und Iteration

- Definition mit freistehenden **eckigen Klammern**

```
a_list = [1, 2, 3]
```

- Eckige Klammern: **Indexoperator** zum Elementzugriff

```
print(a_list[0])    # => 1
```

```
print(a_list[1])    # => 2
```

- Index relativ zum Ende der Liste: **negativer Index**

```
print(a_list[-1])   # => 3
```

```
print(a_list[-2])   # => 2
```

Listen in Python

Definition, Zugriff und Iteration

- Definition mit freistehenden **eckigen Klammern**

```
a_list = [1, 2, 3]
```

- Eckige Klammern: **Indexoperator** zum Elementzugriff

```
print(a_list[0])    # => 1
```

```
print(a_list[1])    # => 2
```

- Index relativ zum Ende der Liste: **negativer Index**

```
print(a_list[-1])   # => 3
```

```
print(a_list[-2])   # => 2
```

- **Iterieren** über Listenelemente

```
for i in a_list:  
    print("%d" % i, end=', ')
```

 # => 1, 2, 3,

Listen in Python

Definition, Zugriff und Iteration

- Definition mit freistehenden **eckigen Klammern**

```
a_list = [1, 2, 3]
```

- Eckige Klammern: **Indexoperator** zum Elementzugriff

```
print(a_list[0])    # => 1
```

```
print(a_list[1])    # => 2
```

- Index relativ zum Ende der Liste: **negativer Index**

```
print(a_list[-1])   # => 3
```

```
print(a_list[-2])   # => 2
```

- **Iterieren** über Listenelemente

```
for i in a_list:
```

```
    print("%d" % i, end=', ')    # => 1, 2, 3,
```

- Typische for-Schleife mit **Laufvariable** kein ideomatisches **Python** – auch bei tatsächlichen „Zählaufgaben“:

```
for i in range(0, 10):
```

```
    print("%d" % i)    # => 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Slices

Stücke aus Listen

- Notation mit `:` im Index-Operator:
`[ lower_boundary : upper_boundary ]`
- Intervalldefinition: `[lowerBoundary; upperBoundary)`
– links inklusive, rechts exklusive
- Notation ermöglicht auch Schrittweite:
`[ lower_boundary : upper_boundary : step ]`

```
another_list = ['a', 'b', 'c', 'd', 'e']  
print(another_list[1:3])      # => ['b', 'c']  
print(another_list[-2:-1])    # => ['d']  
print(another_list[0:-1:2])   # => ['a', 'c']
```

List Comprehensions

Funktionale Listendefinitionen

```
integers      = [i for i in range(0, 5)]  
# => [0, 1, 2, 3, 4]  
squares      = [i**2 for i in range(0, 5)]  
# => [0, 1, 4, 9, 16]  
even_numbers = [i for i in integers  
                if i % 2 == 0]  
# => [0, 2, 4]  
even_squares = [i**2 for i in range(0, 5)  
                if i**2 % 2 == 0]  
# => [0, 4, 16]
```

Python-Funktionen

Ein großer Happen

```
def quicksort(arr):  
    if len(arr) <= 1:  
        return arr  
    pivot = arr[len(arr) // 2]  
    left = [x for x in arr if x < pivot]  
    middle = [x for x in arr if x == pivot]  
    right = [x for x in arr if x > pivot]  
    return quicksort(left) + middle + quicksort(right)  
  
print(quicksort([3,6,8,10,1,2,1]))
```

Klassen, Objekte und Vererbung

OOP in Python in aller Kürze

```
class Answer:
    def __init__(self):
        self._answer = 42

    def compute(self):
        print("The answer to life, the universe "\
              "and everything is %d." % self._answer)

class ConfigurableAnswer(Answer):
    def __init__(self, answer):
        super().__init__()
        self.answer_ = answer

a = Answer(); b = ConfigurableAnswer(42)
a.compute(); b.compute()
```


NumPy

Die Standardbibliothek für Numerik

```
import numpy as np
```

- Hochoptimiert, C/C++/Fortran-Kernbibliotheken
- Im Kontext *Deep Learning* vor allem für lineare Algebra

```
a = np.array([ x**2 for x in range(0, 5) ])
```

```
m = np.array([ [ 1, 2 ],  
               [ 3, 4 ] ],  
              float)
```

```
m[:, 1]          # => array([ 2.,  4.])
```

```
m[1, :]          # => array([ 3.,  4.])
```

```
m.max()          # => 4.0
```

```
m.max(axis=0)     # => array([ 3.,  4.])
```

```
m.max(axis=1)     # => array([ 2.,  4.])
```

Elemente der linearen Algebra

Warum es *TensorFlow* heißt

```
import numpy as np
```

- Liste: `l = np.array([1, 2, 3])`

Elemente der linearen Algebra

Warum es *TensorFlow* heißt

```
import numpy as np
```

- Liste: `l = np.array([1, 2, 3])`
- Vektor

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

```
a = np.array([[1, 2, 3]], float).T  
# => array([[1.], [2.], [3.]])
```

Elemente der linearen Algebra

Warum es *TensorFlow* heißt

```
import numpy as np
```

- Liste: `l = np.array([1, 2, 3])`
- Vektor

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

```
a = np.array([[1, 2, 3]], float).T  
# => array([[1.], [2.], [3.]])
```

- Matrix

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & \dots & A_{1,j} & \dots & A_{1,n} \\ \vdots & & A_{i,j} & & \vdots \\ A_{m,1} & \dots & A_{m,j} & \dots & A_{m,n} \end{bmatrix}$$

```
np.array([ [1, 2], [3, 4] ])  
# => array([[1, 2],  
#           [3, 4]])
```

Elemente der linearen Algebra

Warum es *TensorFlow* heißt

```
import numpy as np
```

- Tensor...„nur“ eine Matrix mit > 2 Achsen

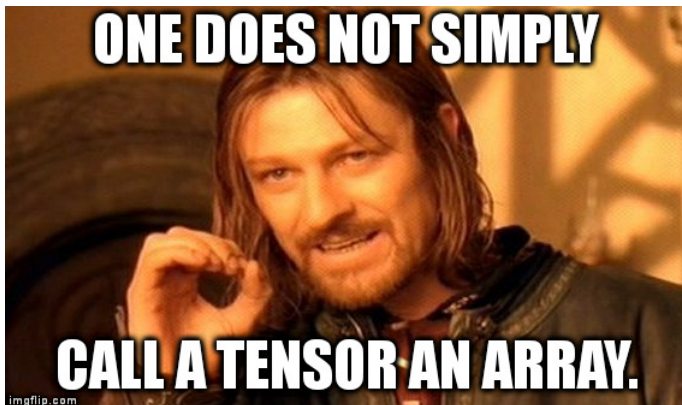
```
t = np.array([[[ 0,  1,  2],  
               [ 3,  4,  5],  
               [ 6,  7,  8]  
              ],  
             [[ 9, 10, 11],  
              [12, 13, 14],  
              [15, 16, 17]  
              ],  
             [[18, 19, 20],  
              [21, 22, 23],  
              [24, 25, 26]]])  
  
t.shape      # => (3, 3, 3)
```

Elemente der linearen Algebra

Warum es *TensorFlow* heißt

```
import numpy as np
```

- Tensor... „nur“ eine Matrix mit > 2 Achsen



... nicht ganz wissenschaftlich korrekt. ;-)

Wichtigste Operatoren (1)

```
import numpy as np
```

■ Transposition

$$\mathbf{x}^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}^T = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$$

```
x = np.array([ 1, 2, 3 ])
x.transpose()
```

$$\left(\mathbf{A}^T\right)_{i,j} = A_{j,i}$$

```
m = np.matrix([[1.0, 2.0],
               [3.0, 4.0]])
m.transpose()
# => matrix([[1., 3.],
#           [2., 4.]])
```

Wichtigste Operatoren (2)

```
import numpy as np
```

■ Skalarmultiplikation

$$a \cdot \mathbf{x} = \begin{bmatrix} a \cdot x_1 & a \cdot x_2 & \dots & a \cdot x_n \end{bmatrix}^T$$

```
x = np.array([ 1.0, 2.0, 3.0 ])
```

```
x * 5
```

```
# => array([ 5.0, 10.0, 15.0 ])
```


Wichtigste Operatoren (3)

Produkte

```
import numpy as np
```

■ Matrix-Vektor-Produkt

$$\mathbf{Ax} = \begin{bmatrix} A_{1,1} & \dots & A_{1,j} & \dots & A_{1,n} \\ \vdots & & A_{i,j} & & \vdots \\ A_{m,1} & \dots & A_{m,j} & \dots & A_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$
$$= \begin{bmatrix} A_{1,1}x_1 & \dots & A_{1,j}x_j & \dots & A_{1,n}x_n \\ \vdots & & A_{i,j}x_j & & \vdots \\ A_{m,1}x_1 & \dots & A_{m,j}x_j & \dots & A_{m,n}x_n \end{bmatrix}$$

```
m = np.array([ [1.0, 1.0], [1.0, 1.0] ])
v = np.array([ 2.0, 2.0 ])
m * v
# => array([ [2.0, 2.0],
#           [2.0, 2.0] ])
```

Wichtigste Operatoren (4)

Produkte

```
from numpy import array
```

■ Matrixprodukt

$$\mathbf{C} = \mathbf{A}\mathbf{B}$$

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

$$\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{n \times p} \rightarrow \mathbf{C} \in \mathbb{R}^{m \times p}$$

```
a = array([[1., 2.], [3., 4.]])
```

```
b = array([[2., 2.], [2., 2.]])
```

```
a @ b
```

```
# => array([ [6.,  6. ],
```

```
#           [14., 14.] ])
```

Wichtigste Operatoren (4)

Produkte

```
from numpy import array
```

■ Hadamard-Produkt

$$\mathbf{C} = \mathbf{A} \odot \mathbf{B}$$

$$C_{i,j} = A_{i,j} \cdot B_{i,j}$$

```
a = array([[1., 2.], [3., 4.]])
```

```
b = array([[2., 2.], [2., 2.]])
```

```
a * b
```

```
# => array([ [2., 4.],  
#           [6., 8.] ])
```

Initialisierungsfunktionen für NumPy-Arrays

- Gefüllt mit **Nullen**

```
np.zeros((2, 2))      # => array([[0., 0.],  
                      #          [0., 0.]])
```

- Gefüllt mit **Einsen**

```
np.ones((2, 2))       # => array([[1., 1.],  
                      #          [1., 1.]])
```

- Gefüllt mit **konstantem Wert**

```
np.full((2, 2), 42.)  # => array([[42., 42.],  
                      #          [42., 42.]])
```

- **Identitätsmatrix**

```
np.eye(2)             # => array([[1., 0.],  
                      #          [0., 1.]])
```

- Mit **Zufallszahlen** gefüllt

```
np.random.random(2)
```

Datenexploration und -vorverarbeitung

Vorverarbeitung...?

Das Wichtigste auf einen Blick

- “Know your data”:
 - Was bedeuten die einzelnen Werte?
 - Sind sie konsistent?
- Fehlende Werte
 - In Ordnung? (= 0)? Abhängig vom Kontext
 - Lineare interpolation
 - State Estimation
 - ANN-basierte Generierung (!)
- Vorverarbeitung
 - Skalieren: $x'_i = \frac{x_i - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} : x_i \mapsto [0; 1] \forall x_i$
 - Bei stationären Daten: Pro Datum Durchschnitt des gesamten Datensatzes subtrahieren
 - „Zentriert“ Daten um den Durchschnitt: Hilft beim Lernen
 - Beispiel Bildverarbeitung: gleichmäßige Helligkeit

Datenexploration

Wenige Tabellen, viele Graphen

```
import h5py
import pandas as pd
import matplotlib.pyplot as plt
```

```
data_file = h5py.File(
    'data/simulation_data.hdf5',
    'r')
p_res = data_file['commercial/P'][(0)]
data = pd.DataFrame(p_res)
```

- Ca. $\frac{1}{3}$ der Daten als Testdaten sofort segmentieren und nicht ansehen (*Data Bias*)

```
In [5]: 1 data.head()
```

Out[5]:

	0
0	609.0
1	603.0
2	599.0
3	614.0
4	606.0

```
In [6]: 1 data.describe()
```

Out[6]:

	0
count	35040.000000
mean	615.043522
std	139.995843
min	413.000000
25%	503.000000
50%	569.000000
75%	715.000000
max	1108.000000

Schöne Graphen mit Matplotlib

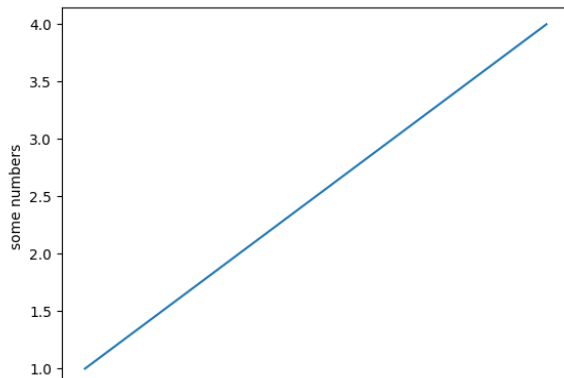
In wenigen Schritten zum schönen Diagramm

```
import matplotlib.pyplot as plt
```

■ Grundfunktion: `plt.plot(·)`

```
plt.plot([1, 2, 3, 4])
```

```
plt.ylabel('some numbers')
```



Schöne Graphen mit Matplotlib

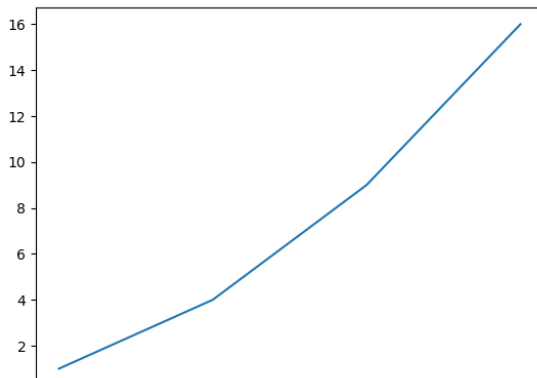
In wenigen Schritten zum schönen Diagramm

```
import matplotlib.pyplot as plt
```

- Grundfunktion: `plt.plot(·)`

- Darstellung von x gegen y

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```



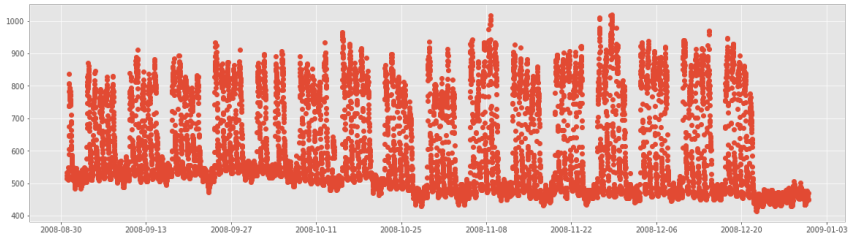
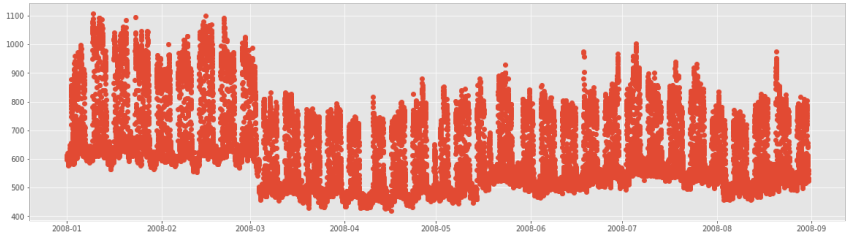
Graphen mit Matplotlib

Vielfältige Darstellungsmöglichkeiten für Notebooks

```
ts = pd.date_range('2008-01-01', periods=len(p_res),
                  freq=pd.DateOffset(minutes=15))
# => DatetimeIndex(['2008-01-01 00:00:00',
                  '2008-01-01 00:15:00', ...])
fig, ax = plt.subplots(2, 1, figsize=(20, 12))
_ = ax[0].plot_date(ts[train_begin * 96:train_end * 96],
                  p_res_train)
_ = ax[1].plot_date(ts[test_begin * 96:test_end * 96],
                  p_res_test)
```

Graphen mit Matplotlib

Vielfältige Darstellungsmöglichkeiten für Notebooks



Umgang mit Datenlücken oder nicht-numerischen Daten

Pandas & Scikit-Learn

- Zeilen mit Lücken entfernen (nicht sinnvoll bei Zeitreihen)
`data.dropna(subset=["my", "attributes"])`
- Das ganze Attribut entfernen:
- Auffüllen – bevorzugt mit dem Median, nicht Durchschnitt (begünstigt Extremwerte)
- Textdaten (z. B. Kategorien) in numerische Daten überführen

Umgang mit Datenlücken oder nicht-numerischen Daten

Pandas & Scikit-Learn

- Zeilen mit Lücken entfernen (nicht sinnvoll bei Zeitreihen)
- Das ganze Attribut entfernen:
`data.drop("attribute", axis=1)`
- Auffüllen – bevorzugt mit dem Median, nicht Durchschnitt (begünstigt Extremwerte)
- Textdaten (z. B. Kategorien) in numerische Daten überführen

Umgang mit Datenlücken oder nicht-numerischen Daten

Pandas & Scikit-Learn

- Zeilen mit Lücken entfernen (nicht sinnvoll bei Zeitreihen)
- Das ganze Attribut entfernen:
- Auffüllen – bevorzugt mit dem Median, nicht Durchschnitt (begünstigt Extremwerte)

```
from sklearn.preprocessing import Imputer
imputer = Imputer(strategy='median')
numerical_data = data.drop("textual_attribute", axis=1)
imputer.fit(numerical_data)
imputed_data = imputer.transform(numerical_data)
```

- Textdaten (z. B. Kategorien) in numerische Daten überführen

Umgang mit Datenlücken oder nicht-numerischen Daten

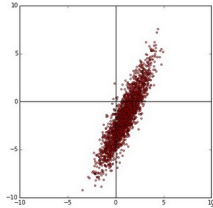
Pandas & Scikit-Learn

- Zeilen mit Lücken entfernen (nicht sinnvoll bei Zeitreihen)
- Das ganze Attribut entfernen:
- Auffüllen – bevorzugt mit dem Median, nicht Durchschnitt (begünstigt Extremwerte)
- Textdaten (z. B. Kategorien) in numerische Daten überführen

```
from sklearn.preprocessing import OneHotEncoder
my_test_cats # => ['PV', 'NO-PV', 'LAND', 'PV', ...]
encoder = OneHotEncoder()
reshaped = my_test_cats.reshape(-1, 1)
            # => array([[ 'PV'], [ 'PV'], ...])
cat_onehot = encoder.fit_transform(reshaped)
            # => array([[ 1.,  0.,  0. ], [0.,  1.,  0.], ...])
```

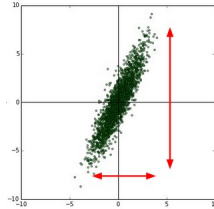
Normalisierung und Zentrierung

Die häufigsten Form der Vorverarbeitung



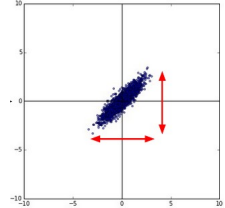
Original

⇒



Nullzentriert

⇒

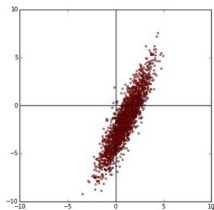


Normalisierung

Stanford CS231n — MIT

Normalisierung und Zentrierung

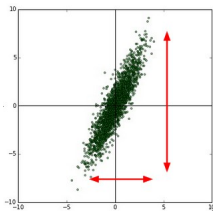
Die häufigsten Form der Vorverarbeitung



Original

X

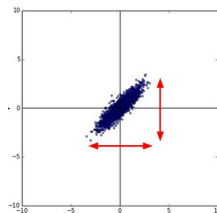
⇒



Nullzentriert

```
X -=  
np.mean(X,  
axis = 0)
```

⇒



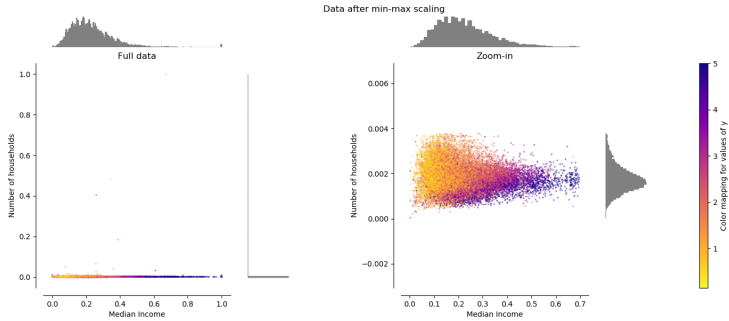
Normalisierung

```
X /= np.std(X,  
axis = 0)
```

Stanford CS231n — MIT

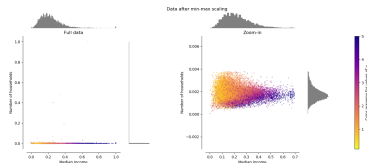
Normalisierung und Zentrierung

Mit *scikit-learn*



Normalisierung und Zentrierung

Mit *scikit-learn*

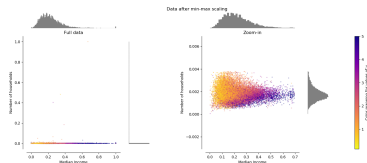


```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
```

```
data = np.array([ 1., 2., 3., 4. ])
data = data.reshape(-1, 1)
# => array([ [1.], [2.], [3.], [4.] ])
scaler = sklearn.preprocessing.MinMaxScaler()
scaled_data = scaler.fit_transform(data)
# => array([[0], [0.33], [0.666], [1.]])
```

Normalisierung und Zentrierung

Mit *scikit-learn*



```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
```

```
data = np.array([ 1., 2., 3., 4. ])
data = data.reshape(-1, 1)
# => array([ [1.], [2.], [3.], [4.] ])
scaler = sklearn.preprocessing.MinMaxScaler()
scaled_data = scaler.fit_transform(data)
# => array([[0], [0.33], [0.666], [1.]])
```

Normalisierung und Zentrierung

Mit *scikit-learn*

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

data = np.array([[ 1, 2, 3, 4 ],
                 [ 4, 5, 2, 1 ],
                 [ 9, 7, 3, 2 ]], float)

scaler = MinMaxScaler()
scaler.fit(data)
scaled = scaler.transform(data)
# => array([[ 0.    ,  0.    ,  1.    ,  1.    ],
#          [ 0.375,  0.6   ,  0.    ,  0.    ],
#          [ 1.    ,  1.    ,  1.    ,  0.33333333]])
std = (data - data.min(axis=0))
      / (data.max(axis=0) - data.min(axis=0))
scaled = x_std * (max - min) + min

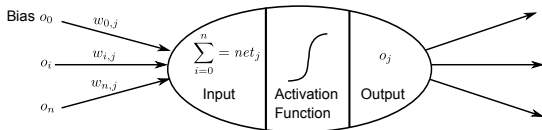
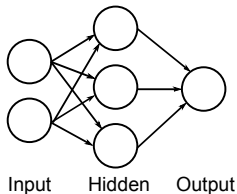
scaler.inverse_transform(scaled)
```

Künstliche Neuronale Netze

Artificial Neural Networks (ANNs)

Das Perceptron

Was ist eigentlich ein künstliches neuronales Netz?

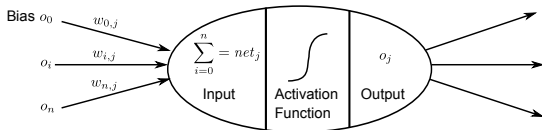
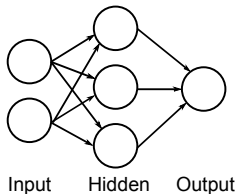


Bestandteile eines ANN:

- **Neuronen mit Aktivierungsfunktionen** (nicht-linear!)
- **gewichtete Verbindungen** zwischen Neuronen: Parameter θ

Das Perceptron

Was ist eigentlich ein künstliches neuronales Netz?

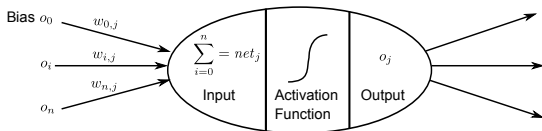
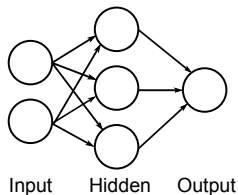


Skalarnotation:

$$a_i^{(l)} = f \left(\sum_j w_{i,j}^{(l)} a_j^{(l-1)} + b_i^{(l)} \right)$$

Das Perceptron

Was ist eigentlich ein künstliches neuronales Netz?



Skalarnotation:

$$a_i^{(l)} = f \left(\sum_j w_{i,j}^{(l)} a_j^{(l-1)} + b_i^{(l)} \right)$$

Vektornotation:

$$\mathbf{a}^{(l)} = F \left(\mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \right)$$

Single Instruction, Multiple Data (SIMD): Kernkonzept aktueller GPGPU-beschleunigter Verfahren

Aktivierungsfunktionen

Das neuronale Herzstück

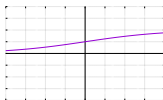
Identität



$$f(x) = x$$

$$f'(x) = 1$$

Sigmoid



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

tanh



$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

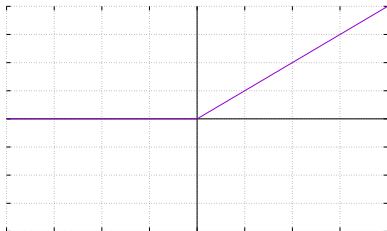
$$\tanh'(x) = 1 - \tanh^2(x)$$

- ... Und mehr: Softplus, hard tanh, etc.
- **Spezialfälle**

ReLU

Die erste Wahl für Regression

$$f(x) = \begin{cases} 0 & \text{für } x < 0, \\ x & \text{für } x \geq 0 \end{cases}$$
$$f'(x) = \begin{cases} 0 & \text{für } x < 0, \\ 1 & \text{für } x \geq 0 \end{cases}$$



- Nichtlinear
- Saturiert nicht (cf. $f'(x)$)
- **konsistenten Gradient**: einfach zu optimieren
- $f''(x) = 0$ fast überall: sehr gut für **gradientenbasierte Trainingsalgorithmen**

SoftMax

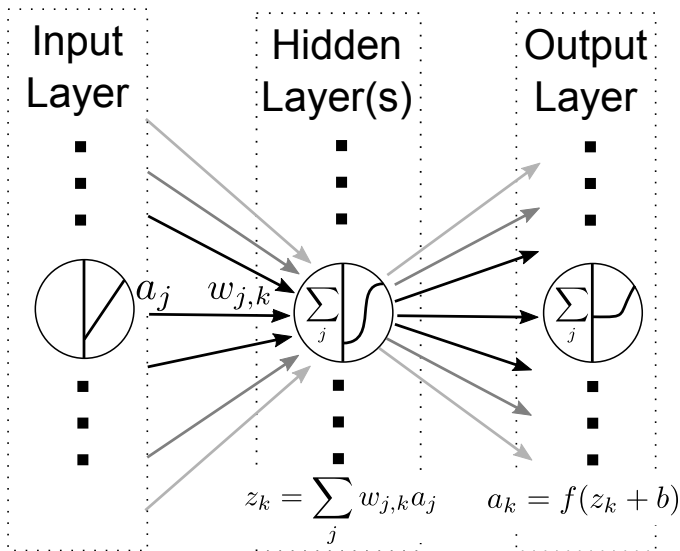
Für Klassifizierungsaufgaben

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

- Nichtlinear
- Repräsentiert Kategorien
- Wahrscheinlichkeit, dass eine Eingabe in eine Kategorie fällt
- Summe über alle Kategorien: 1.0

Was ist nun ein künstliches neuronales Netz?

Die graphische Erklärung



Was ist nun ein künstliches neuronales Netz?

Statistische Modelle

- Jedes ANN ist ein **Approximator**
- Ein **statistisches Modell** zur Vorhersage eines Problems:

$$\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$$

Wieviele Darstellungen für $f(x)$ existieren, so dass gilt. . .

- $f(5) = 2$

Was ist nun ein künstliches neuronales Netz?

Statistische Modelle

- Jedes ANN ist ein **Approximator**
- Ein **statistisches Modell** zur Vorhersage eines Problems:

$$y = f(x; \theta)$$

Wieviele Darstellungen für $f(x)$ existieren, so dass gilt. . .

- $f(5) = 2$
 - $f(x) = x - 3$

Was ist nun ein künstliches neuronales Netz?

Statistische Modelle

- Jedes ANN ist ein **Approximator**
- Ein **statistisches Modell** zur Vorhersage eines Problems:

$$y = f(x; \theta)$$

Wieviele Darstellungen für $f(x)$ existieren, so dass gilt. . .

- $f(5) = 2$
 - $f(x) = x - 3$
- $f(4) = 2$
 - $f(x) = \lfloor \frac{x}{2} \rfloor$

Was ist nun ein künstliches neuronales Netz?

Statistische Modelle

- Jedes ANN ist ein **Approximator**
- Ein **statistisches Modell** zur Vorhersage eines Problems:

$$y = f(x; \theta)$$

Wieviele Darstellungen für $f(x)$ existieren, so dass gilt. . .

- $f(5) = 2$
 - $f(x) = x - 3$
 - $f(4) = 2$
 - $f(x) = \lfloor \frac{x}{2} \rfloor$
 - $f(x) = 2$
- ...?

Was ist nun ein künstliches neuronales Netz?

Statistische Modelle

- Jedes ANN ist ein **Approximator**
- Ein **statistisches Modell** zur Vorhersage eines Problems:

$$y = f(x; \theta)$$

Wieviele Darstellungen für $f(x)$ existieren, so dass gilt. . .

- $f(5) = 2$
 - $f(x) = x - 3$
- $f(4) = 2$
 - $f(x) = \lfloor \frac{x}{2} \rfloor$
 - $f(x) = 2$
- ...?
- Ergebnis: **unendlich viele**.

Das ANN als statistisches Modell

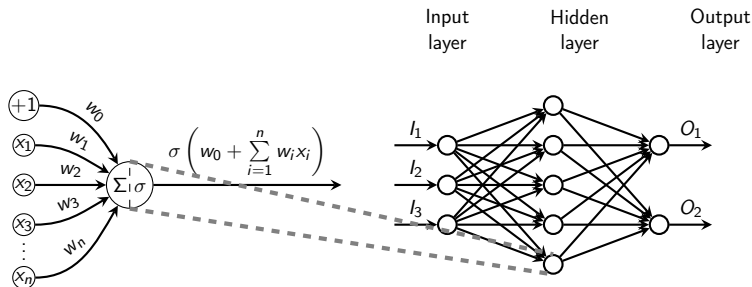
- Mathematische Eingrenzung: **Kontinuität** und **Beschränkung**
(There are many ways to represent a function)
- Netze aufgrund der **Funktionskomposition**:

$$f(\mathbf{x}; \boldsymbol{\theta}) = f^{(n)}(f^{(n-1)}(\dots f^{(2)}(f^{(1)}(\mathbf{x}; \boldsymbol{\theta}))))$$

- $1, 2, \dots, n$: Schichten (**Layer**)
Mehr Schichten $\approx$ mehr Merkmale
- Jedes Modell hat eine **Kapazität** (cf. Informationstheorie)

Grenzen des Perceptrons

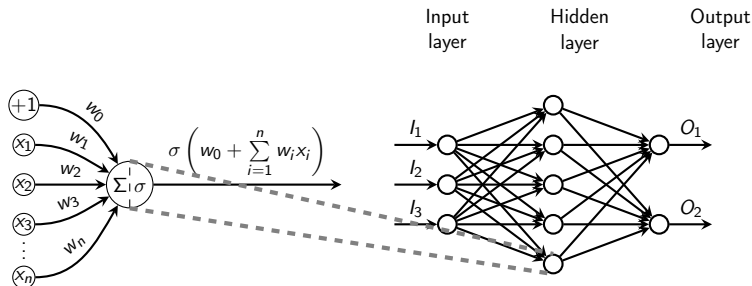
Warum weiterentwickeln?



„Wenn ein mehrlagiges Perceptron (Multi-Layer Perceptron, MLP) jede Funktion approximieren kann – wozu brauchen wir dann andere Architekturen?“

Grenzen des Perceptrons

Warum weiterentwickeln?



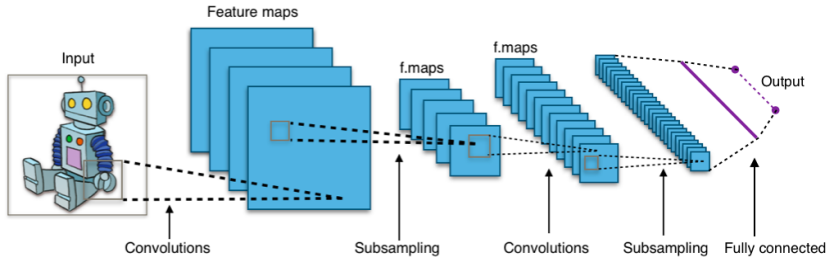
„Wenn ein mehrlagiges Perceptron (Multi-Layer Perceptron, MLP) jede Funktion approximieren kann – wozu brauchen wir dann andere Architekturen?“

— Weil wir keine perfekten Trainingsalgorithmen und auch nicht unendlich viel Zeit für das Training haben.

Muster- und Objekterkennung

Convolutional Neural Networks (CNNs, ConvNets)

Struktur

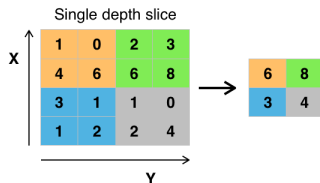
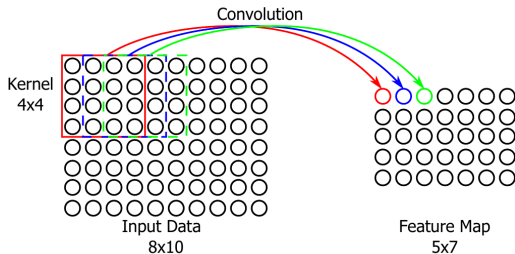


Aphex34 — CC-BY-SA 4.0

- Architektur als intelligente Abkürzung
- Inspiriert vom menschlichen Sehen

Convolutional Neural Networks (CNNs, ConvNets)

Convolution and Pooling



Aphex34 — CC-BY-SA 4.0

Convolution

- Diskrete Faltung: Lokale Umgebung
- 2D-/3D-Anordnung der Neuronen

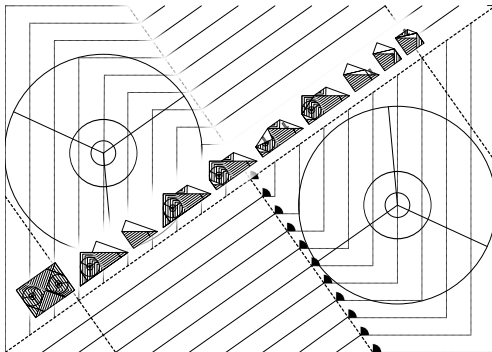
Maxpooling

- Stärkstes Merkmal selektieren (bspw. Kante)

Faltung...?

Kurze Erklärung einer mathematischen Operation

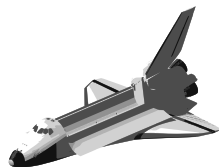
■ Faltung...?



Faltung...?

Kurze Erklärung einer mathematischen Operation

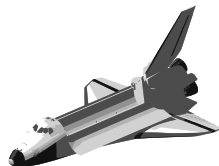
- Verfolgen eines Raumschiffs mit einem Lasersensor: $x(t)$ Position zum Zeitpunkt t



Faltung...?

Kurze Erklärung einer mathematischen Operation

- Verfolgen eines Raumschiffs mit einem Lasersensor: $x(t)$ Position zum Zeitpunkt t
- Laser rauscht; Durchschnitt mehrerer Messungen (gewichteter Mittelwert): $w(a)$ einer a alten Messung



Faltung...?

Kurze Erklärung einer mathematischen Operation

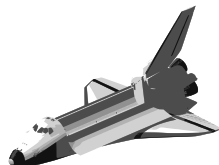
- Verfolgen eines Raumschiffs mit einem Lasersensor: $x(t)$ Position zum Zeitpunkt t
- Laser rauscht; Durchschnitt mehrerer Messungen (gewichteter Mittelwert): $w(a)$ einer a alten Messung
- Kombination zweier Funktionen:

$$s(t) = \int x(t)w(t - a)$$

Feature Map

Eingabe

Kernel



Faltung...?

Kurze Erklärung einer mathematischen Operation

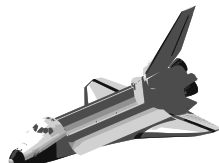
- Verfolgen eines Raumschiffs mit einem Lasersensor: $x(t)$ Position zum Zeitpunkt t
- Laser rauscht; Durchschnitt mehrerer Messungen (gewichteter Mittelwert): $w(a)$ einer a alten Messung
- Kombination zweier Funktionen:

$$s(t) = \int x(t)w(t-a)$$

$$s(t) = (x * w)(t)$$

- Computer arbeiten **wertdiskret**:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$



Faltung...?

Kurze Erklärung einer mathematischen Operation

- Kombination zweier Funktionen:

$$s(t) = \int x(t)w(t-a)$$

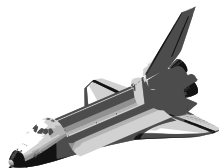
- Computer arbeiten **wertdiskret**:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- 2D (Bilder):

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n)$$

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m,j-n)K(m,n)$$

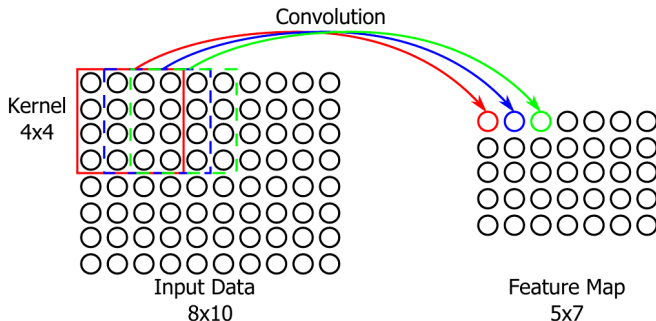


Zweck der Faltung

Eine architekturelle Abkürzung

■ 2D-Faltung (Bilder):

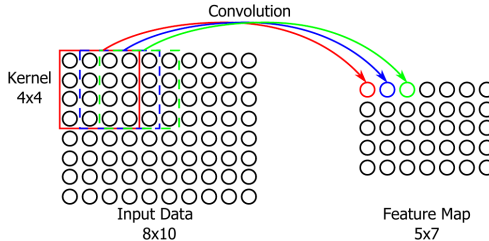
$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n)$$



Zweck der Faltung

Eine architekturelle Abkürzung

■ Vorteile:



- **Sparse weights** – dünn besetzte Gewichtsmatrix
- **Parameter sharing** – Funktionen (Bildausschnitte) teilen sich Parameter
- **Equivariant representations** – Verschiebung in Eingabedaten verschiebt Ausgabe im selben Maße

In die Praxis – Objekterkennung: PV-Anlagen auf Hausdächern

Was steckt hinter *Google Sunroof*?

- **Aufgabe:** Häuser ohne PV-Anlage identifizieren
- **Klassifizierungsaufgabe:** (PV, No-PV, Land)
- **Schnittstelle:** Daten liegen vor (bspw. Grundstücksgrenzen aus OpenStreetMap, Photos aus Google Maps)



PV



No PV



Land

PV Rooftop Detection

Schritt 1: Imports und Bilder laden

```
import os
import numpy as np
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten
from keras.preprocessing.image import ImageDataGenerator, load_img

img_generator = ImageDataGenerator(data_format='channels_first',
                                   featurewise_center=True,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   rotation_range=20,
                                   shear_range=2.0,
                                   fill_mode='nearest')

train_data = img_generator.flow_from_directory(DATA_PATH, target_s
```

PV Rooftop Detection

Netzaufbau

- Bildverarbeitung: Conv2D, MaxPooling2D
- Klassifizierung: Softmax-Schicht am Ende
- Gegen Overfitting: Dropout

```
model = Sequential()
model.add(Conv2D(64, activation='relu',
                 input_shape=(3, 128, 128),
                 kernel_size=(3, 3),
                 data_format='channels_first'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(28, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(len(train_data.class_indices),
                 activation='softmax'))
```

PV Rooftop Detection

Test des Lernerfolgs

```
test_images = ('test-pv', 'test-no-pv', 'test-land')
def predict_image(img_name):
    img_path = "data/%s.png" % img_name
    img = load_img(img_path, target_size=(128, 128),
                   color_mode='rgb')
    data = img_to_array(img).T
    data = np.expand_dims(data, axis=0)
    print(np.shape(data))
    return model.predict(data)
dict(zip(test_images,
         [ predict_image(name) for name in test_images ]))
# => {'test-pv': [0.0, 0.0, 1.0]}
#     {'test-no-pv': [0.0, 1.0, 0.0]}
#     {'test-land': [1.0, 0.0, 0.0]}
```

- Achtung: Klares **Overfitting** im Mini-Beispiel!

In die Praxis – Objekterkennung: PV-Anlagen auf Hausdächern

Was steckt hinter *Google Sunroof*?

jupyter PV-Rooftop-Detection Last Checkpoint: vor 2 Stunden (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Bildtransformationen anwendet. Aber auch andere Daten wie Zeitreihen können variiert werden -- im Zweifel durch Hinzufügen einfachen Gausschen Rauschens.

Original	Rotieren	Verzerren	Spiegeln
	 	 	 

Training künstlicher neuronaler Netze

Verzerrung, Varianz und Regularisierung

Was heißt „Training“?

Arten des Lernens

■ Aufgabe (Task):

- Klassifikation: $f : \mathbb{R}^n \rightarrow 1, \dots, k$
- Regression: $f : \mathbb{R}^n \rightarrow \mathbb{R}$
- Imputation fehlender Werte: $\mathbf{x} \in \mathbb{R}^n$ $\mathbf{x}$ dünn besetzt
- Transkription, Übersetzung, Anomalieerkennung, ...

■ Leistungskennwerte (Performance Measure):

- Genauigkeit
- Fehler

■ Erfahrung (Experience):

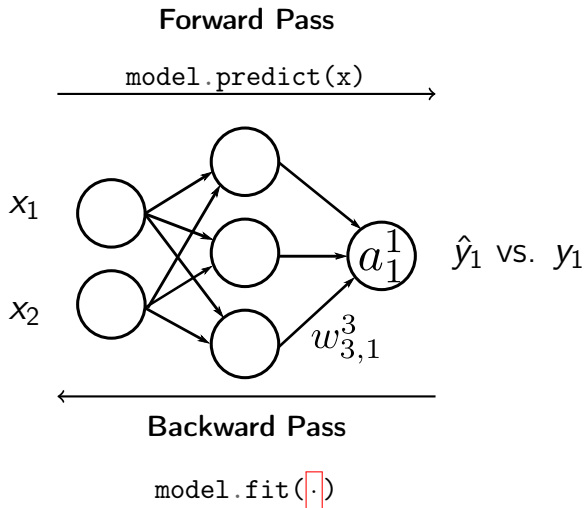
- Überwacht
- Unüberwacht

■ Quintessenz: θ basierend auf bekannten Daten ermitteln (und dabei an den Elefanten denken...)

- Training set
- Test set
- Validation set

Backpropagation of Error

Das wichtigste Trainingsverfahren für ANNs



Hyperparameter

Viele Stellschrauben neben der eigentlichen Netzstruktur

```
from keras.model import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import SGD
```

```
model = Sequential()
model.add(Dense(4))
model.add(Dense(45))
model.add(Dropout(0.5))
model.add(Dense(2))
```

```
optimizer = SGD(lr=learning_rate)
model.compile(optimizer, loss='mse')
model.fit(x_feature_data, y_label_data,
          epochs=ep,
          batch_size=bs)
```

Hyperparameter

Viele Stellschrauben neben der eigentlichen Netzstruktur

```
from keras.model import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import SGD
```

```
model = Sequential()
model.add(Dense(4))
model.add(Dense(45))
model.add(Dropout(0.5))
model.add(Dense(2))
```

```
optimizer = SGD(lr=learning_rate)
model.compile(optimizer, loss='mse')
model.fit(x_feature_data, y_label_data,
          epochs=ep,
          batch_size=bs)
```

Dropout-Wahrscheinlichkeit

Trainingsverfahren

Lernrate

Kostenfunktion

Trainingsepochen

Batch Size

Backpropagation of Error

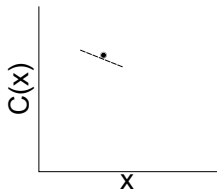
Grundlegende Annahmen

- Bekannte Größen bei **überwachtem Lernen**:
 - Ausgabe des ANN ($\hat{y} = h(\mathbf{x})$)
 - Gewünschte Ausgabe ($\mathbf{y}$)
- Strategie:
 1. Den Fehler des Durchlaufs ermitteln
 2. Beitrag jedes Gewichts ermitteln
 3. Gewichte (**Parameter**) anpassen
- Fehlerfunktion unbekannt: Beitrag einzelner Gewichte nicht analytisch feststellbar

Backpropagation of Error

Grundlegende Annahmen

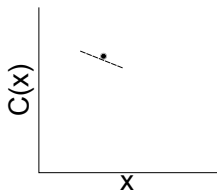
- Bekannte Größen bei **überwachtem Lernen**:
 - Ausgabe des ANN ($\hat{y} = h(x)$)
 - Gewünschte Ausgabe (y)
- Strategie:
 1. Den Fehler des Durchlaufs ermitteln
 2. Beitrag jedes Gewichts ermitteln
 3. Gewichte (**Parameter**) anpassen
- Fehlerfunktion unbekannt: Beitrag einzelner Gewichte nicht analytisch feststellbar



Backpropagation of Error

Grundlegende Annahmen

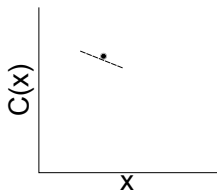
- Bekannte Größen bei **überwachtem Lernen**:
 - Ausgabe des ANN ($\hat{y} = h(\mathbf{x})$)
 - Gewünschte Ausgabe ($\mathbf{y}$)
- Strategie:
 1. Den Fehler des Durchlaufs ermitteln
 2. Beitrag jedes Gewichts ermitteln
 3. Gewichte (**Parameter**) anpassen
- Fehlerfunktion unbekannt: Beitrag einzelner Gewichte nicht analytisch feststellbar
- Ermittelbar: der **Gradient** (ANN: ein Funktionsapproximator)



Backpropagation of Error

Grundlegende Annahmen

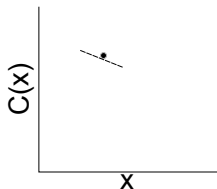
- Bekannte Größen bei **überwachtem Lernen**:
 - Ausgabe des ANN ($\hat{y} = h(x)$)
 - Gewünschte Ausgabe (y)
- Strategie:
 1. Den Fehler des Durchlaufs ermitteln
 2. Beitrag jedes Gewichts ermitteln
 3. Gewichte (**Parameter**) anpassen
- Fehlerfunktion unbekannt: Beitrag einzelner Gewichte nicht analytisch feststellbar
- Ermittelbar: der **Gradient** (ANN: ein Funktionsapproximator)



Backpropagation of Error

Grundlegende Annahmen

- Bekannte Größen bei **überwachtem Lernen**:
 - Ausgabe des ANN ($\hat{y} = h(x)$)
 - Gewünschte Ausgabe (y)
- Strategie:
 1. Den Fehler des Durchlaufs ermitteln
 2. Beitrag jedes Gewichts ermitteln
 3. Gewichte (**Parameter**) anpassen
- Fehlerfunktion unbekannt: Beitrag einzelner Gewichte nicht analytisch feststellbar
- Ermittelbar: der **Gradient** (ANN: ein Funktionsapproximator)



$C(x)$: Kostenfunktion

$$C = \frac{1}{2n} \sum_i (y_i - \hat{y}_i)^2$$

Der Gradient

- Einzelteile des Gradienten: Gewichte & Fehlerwerte
- Einfluss eines Gewichts auf den Fehler
- Gradient einer „inneren“ Verbindung (z. B. Input $\rightarrow$ Hidden)?

Kettenregel:

$$f : \mathbb{R} \mapsto \mathbb{R}$$

$$z = f(y) = f(g(x))$$

$$g : \mathbb{R} \mapsto \mathbb{R}$$

$$y = g(x) ,$$

Der Gradient

- Einzelteile des Gradienten: Gewichte & Fehlerwerte
- Einfluss eines Gewichts auf den Fehler
- Gradient einer „inneren“ Verbindung (z. B. Input $\rightarrow$ Hidden)?

Kettenregel:

$$f : \mathbb{R} \mapsto \mathbb{R}$$

$$z = f(y) = f(g(x))$$

$$g : \mathbb{R} \mapsto \mathbb{R}$$

$$y = g(x) ,$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} .$$

Der Gradient

- Einzelteile des Gradienten: Gewichte & Fehlerwerte
- Einfluss eines Gewichts auf den Fehler
- Gradient einer „inneren“ Verbindung (z. B. Input $\rightarrow$ Hidden)?
Kettenregel:

$$f : \mathbb{R} \mapsto \mathbb{R}$$

$$z = f(y) = f(g(x))$$

$$g : \mathbb{R} \mapsto \mathbb{R}$$

$$y = g(x) ,$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} .$$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \quad \text{für } \mathbf{y} = g(\mathbf{x}) , z = f(\mathbf{y}) ,$$

Der Gradient

- Einzelteile des Gradienten: Gewichte & Fehlerwerte
- Einfluss eines Gewichts auf den Fehler
- Gradient einer „inneren“ Verbindung (z. B. Input $\rightarrow$ Hidden)?
Kettenregel:

$$f : \mathbb{R} \mapsto \mathbb{R}$$

$$z = f(y) = f(g(x))$$

$$g : \mathbb{R} \mapsto \mathbb{R}$$

$$y = g(x) ,$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} .$$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \quad \text{für } \mathbf{y} = g(\mathbf{x}) , z = f(\mathbf{y}) ,$$

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^{\top} \nabla_{\mathbf{y}} z .$$

Vom Gradienten zu Backpropagation of Error

Fehler an einem Neuron

- Fehler eines Neurons abhängig von seiner Eingabe:

$$\delta_j^{(l)} \equiv \frac{\partial C}{\partial z_j} ,$$

- Eingabe: Ausgaben vorgelagerter Neuronen

Vom Gradienten zu Backpropagation of Error

Fehler an einem Neuron

- Fehler eines Neurons abhängig von seiner Eingabe:

$$\delta_j^{(l)} \equiv \frac{\partial C}{\partial z_j} ,$$

- Eingabe: Ausgaben vorgelagerter Neuronen gewichtet

Vom Gradienten zu Backpropagation of Error

Fehler an einem Neuron

- Fehler eines Neurons abhängig von seiner Eingabe:

$$\delta_j^{(l)} \equiv \frac{\partial C}{\partial z_j} ,$$

- Eingabe: Ausgaben vorgelagerter Neuronen gewichtet aufsummiert

Vom Gradienten zu Backpropagation of Error

Fehler an einem Neuron

- Fehler eines Neurons abhängig von seiner Eingabe:

$$\delta_j^{(l)} \equiv \frac{\partial C}{\partial z_j} ,$$

- Eingabe: Ausgaben vorgelagerter Neuronen gewichtet aufsummiert
- Backpropagation: Fehler „von hinten“ propagieren
- **Ausgabeschicht:**

$$\delta_j^{(L)} = \frac{\partial C}{\partial a_j^{(L)}} f'(z_j^{(L)}) \qquad \delta^{(L)} = \nabla_a \mathbf{C} \odot F'(\mathbf{z}^{(L)})$$

- **Übrige Schichten und Bias:**

$$\delta^{(l)} = \left\{ (\mathbf{W}^{(l+1)})^\top \boldsymbol{\delta}^{(l+1)} \right\} \odot F'(\mathbf{z}^{(l)}) \qquad \delta_j^{(l)} = \frac{\partial C}{\partial b_j^{(l)}}$$

Vom Gradienten zu Backpropagation of Error

Fehler an einem Neuron

- Fehler eines Neurons abhängig von seiner Eingabe:

$$\delta_j^{(l)} \equiv \frac{\partial C}{\partial z_j}, \quad \leftarrow \text{Kostenfunktion}$$

- Eingabe: Ausgaben vorgelagerter Neuronen gewichtet aufsummiert
- Backpropagation: Fehler „von hinten“ propagieren
- **Ausgabeschicht:**

$$\delta_j^{(L)} = \frac{\partial C}{\partial a_j^{(L)}} f'(z_j^{(L)}) \qquad \delta^{(L)} = \nabla_a \mathbf{C} \odot F'(\mathbf{z}^{(L)})$$

- **Übrige Schichten und Bias:**

$$\delta^{(l)} = \left\{ (\mathbf{W}^{(l+1)})^\top \delta^{(l+1)} \right\} \odot F'(\mathbf{z}^{(l)}) \qquad \delta_j^{(l)} = \frac{\partial C}{\partial b_j^{(l)}}$$

Backpropagation of Error

Der Algorithmus

1. Eingabe der Trainingsmuster
2. Für jedes Trainingsmuster $\mathbf{x}$:
 - 2.1 **Feedforward:** Für $l = 2, 3, \dots, L$ berechne

$$\mathbf{z}^{\mathbf{x},(l)} = \mathbf{W}^{(l)} \mathbf{a}^{\mathbf{x},(l-1)} + \mathbf{b}^{(l)} \quad \mathbf{a}^{\mathbf{x},(l)} = F(\mathbf{z}^{\mathbf{x},(l)})$$

- 2.2 **Ausgabefehler:**

$$\delta^{\mathbf{x},(l)} = \nabla_{\mathbf{a}} \mathbf{C}_{\mathbf{x}} \odot F'(\mathbf{z}^{\mathbf{x},(L)})$$

- 2.3 **Backpropagation:** Für $l = L - 1, L - 2, \dots, 2$ berechne:

$$\delta^{\mathbf{x},(l)} = \left\{ (\mathbf{W}^{(l+1)})^{\top} \delta^{\mathbf{x},(l+1)} \right\} \odot F'(\mathbf{z}^{\mathbf{x},(l)})$$

3. **Gradientenabstieg:** $l = L, L - 1, \dots, 2$ berechne

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \frac{\eta}{m} \sum_{\mathbf{x}} \delta^{\mathbf{x},(l)} \left(\mathbf{a}^{\mathbf{x},(l-1)} \right)^{\top}$$

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \frac{\eta}{m} \sum_{\mathbf{x}} \delta^{\mathbf{x},(l)}$$

Backpropagation of Error

Der Algorithmus

1. Eingabe der Trainingsmuster
2. Für jedes Trainingsmuster $\mathbf{x}$:
 - 2.1 **Feedforward:** Für $l = 2, 3, \dots, L$ berechne

$$\mathbf{z}^{\mathbf{x},(l)} = \mathbf{W}^{(l)} \mathbf{a}^{\mathbf{x},(l-1)} + \mathbf{b}^{(l)} \quad \mathbf{a}^{\mathbf{x},(l)} = F(\mathbf{z}^{\mathbf{x},(l)})$$

- 2.2 **Ausgabefehler:**

$$\delta^{\mathbf{x},(l)} = \nabla_a \mathbf{C}_x \odot F'(\mathbf{z}^{\mathbf{x},(L)})$$

- 2.3 **Backpropagation:** Für $l = L - 1, L - 2, \dots, 2$ berechne:

$$\delta^{\mathbf{x},(l)} = \left\{ (\mathbf{W}^{(l+1)})^\top \delta^{\mathbf{x},(l+1)} \right\} \odot F'(\mathbf{z}^{\mathbf{x},(l)})$$

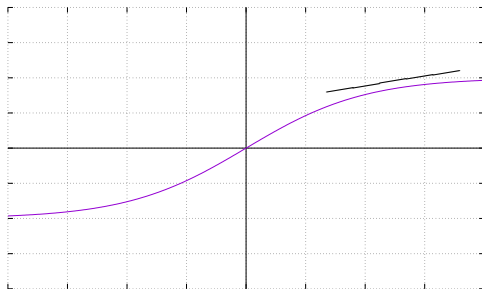
3. **Gradientenabstieg:** $l = L, L - 1, \dots, 2$ berechne

$$\begin{aligned} \mathbf{W}^{(l)} &= \mathbf{W}^{(l)} - \frac{\eta}{m} \sum_{\mathbf{x}} \delta^{\mathbf{x},(l)} \left(\mathbf{a}^{\mathbf{x},(l-1)} \right)^\top \\ \mathbf{b}^{(l)} &= \mathbf{b}^{(l)} - \frac{\eta}{m} \sum_{\mathbf{x}} \delta^{\mathbf{x},(l)} \end{aligned}$$

Learning Rate

Aktivierungsfunktion

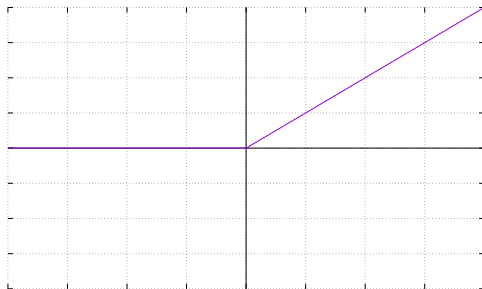
Die passende für Backpropagation of Error



- Verschwindender Gradient: **kein Training**

Aktivierungsfunktion

Die passende für Backpropagation of Error

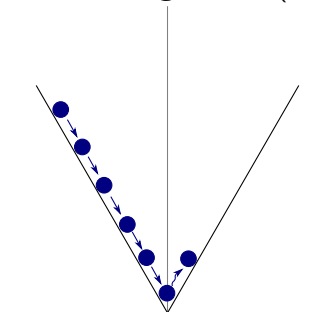


- Nichtlinear
- Saturiert nicht (cf. $f'(x)$)
- **konsistententer Gradient:** einfach zu optimieren
- $f''(x) = 0$ fast überall: sehr gut für **gradientenbasierte Trainingsalgorithmen**

Fehlerfunktionen

Für Regressionen

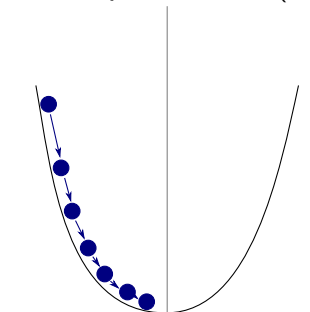
Mean Average Error (ℓ_1)



$$mae = \frac{1}{m} \sum_{i=1}^m |y_i - \hat{y}_i|$$

Robust gegen Ausreißer
Gleicher Gradient überall

Mean Squared Error (ℓ_2)

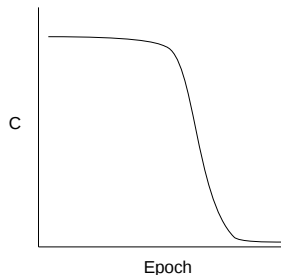
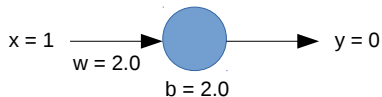
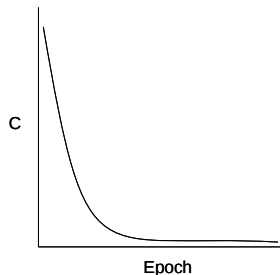
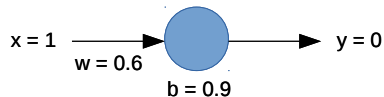


$$mse = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

Variabler Gradient
Anfällig gegen Ausreißer

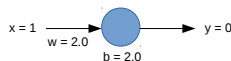
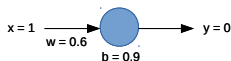
Einfluss der Fehlerfunktion

Das Problem der verschwindenden Gradienten



Einfluss der Fehlerfunktion

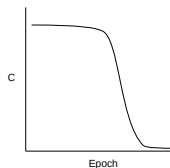
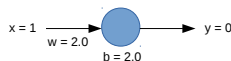
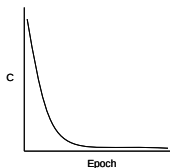
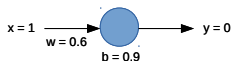
Das Problem der verschwindenden Gradienten



- Bisweilen keine Alternative zu tanh
- Extremwerte (großer Fehler): kleiner Gradient
- Lösungsansatz: Initialisierung des Netzes
 - Möglich, häufig praktiziert
 - Gaussverteilung mit Summe 1.0 pro Schicht
 - Je nach Trainingsdaten trotzdem kleine/sehr große Gradienten

Einfluss der Fehlerfunktion

Das Problem der verschwindenden Gradienten



■ Lösungsansatz: Initialisierung des Netzes

■ Alternative: Andere Fehlerfunktion

- ☐ Fehler sind einheitenlos; keine Vorgabe über die Gestalt der Funktion
- ☐ Anforderung: $C > 0$
- ☐ Anforderung: $\hat{\mathbf{y}} \rightarrow \mathbf{y} : C \rightarrow 0$

Kreuzentropie

Die Standard-Fehlerfunktion für Klassifikation

$$C = -\frac{1}{n} \sum_x [y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})]$$

- Herkunft: Maß für die Qualität eines **Informationsmodells** hinsichtlich einer Wahrscheinlichkeitsverteilung
ANN: Modell eines Problems!

Kreuzentropie

Die Standard-Fehlerfunktion für Klassifikation

$$C = -\frac{1}{n} \sum_x [y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})]$$

■ Mit $a = \sigma(z) = \frac{1}{1+e^{-z}}$:

$$\begin{aligned} \frac{\partial C}{\partial w_j} &= -\frac{1}{n} \sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)} \right) \frac{\partial \sigma}{\partial w_j} \\ &= -\frac{1}{n} \sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)} \right) \sigma'(z) x_j \\ &= \frac{1}{n} \sum_x \frac{\sigma'(z) x_j}{\sigma(z)(1-\sigma(z))} (\sigma(z) - y) \\ &= \frac{1}{n} \sum_x x_j (\sigma(z) - y) \end{aligned}$$

Kreuzentropie

Die Standard-Fehlerfunktion für Klassifikation

$$C = -\frac{1}{n} \sum_x [y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})]$$

- Herkunft: Maß für die Qualität eines **Informationsmodells** hinsichtlich einer Wahrscheinlichkeitsverteilung
ANN: Modell eines Problems!
- Allgemein mit $a = f(z)$:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j (f(z) - y)$$

- Keine Ableitung $f'(z)$ mehr – gleichmäßige Gradienten!
- Kreuzentropie benötigt *Wahrscheinlichkeitsverteilung* über **y** ($\sum_i p(y_i) = 1$) – **unpassend für Regression**, aber unschlagbar für Klassifikation mit SoftMax in der Ausgabeschicht

Batch Size

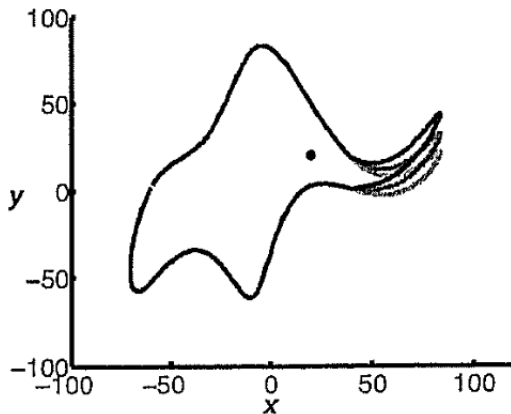
Effizienz versus Genauigkeit

```
model.compile(optimizer='sgd',  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])  
model.fit(train_x, train_y, batch_size=32)
```

- *Batches* effizienter
 - Erlaubt Matrix-Matrix-Multiplikationen
 - Gradient wird pro *Batch* berechnet
 - Tensor: (*batchsize*, $|\mathbf{x}|$)
- Gradientenabstieg mit Änderung pro Muster robuster
- Kompromiss: Zufällig gewählte *Mini Batches* (*Stochastic Gradient Decent*)
- Guter *Batch Size*-Startwert: 32

Overfitting

Mehr Parameter machen kein besseres Modell

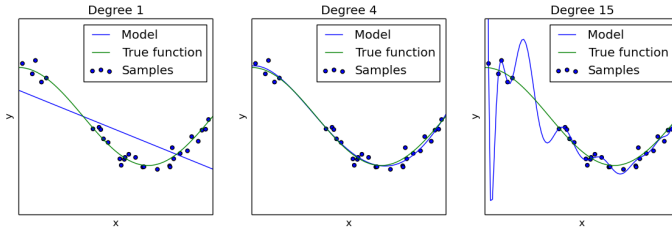


*With **four** parameters, I can draw an elephant,
and with **five**, I can make ihm wiggle his trunk.*

— John von Neumann

Overfitting

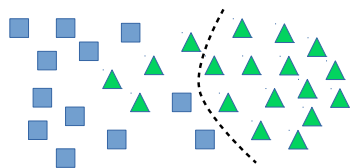
Wenn das Training zu gut läuft...



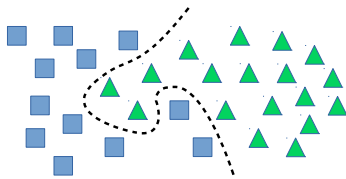
Strong Bias Training mittelmäßig, Test gut
Netz lernt nicht ausreichend gut

Strong Variance Training sehr gut, Test sehr schlecht
Netz lernt statistisches Rauschen

Das Verzerrung-Varianz-Dilemma



hohe **Verzerrung**
Bias



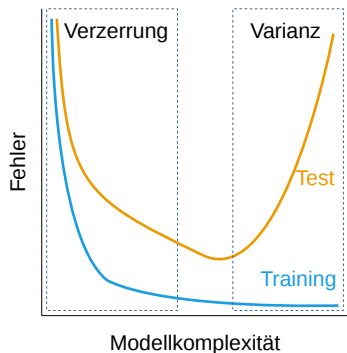
hohe **Varianz**
Variance

Verzerrung Zu wenige Modellparameter, zu komplexe Muster

Varianz Zu viele Modellparameter, zu wenig Muster im Training

Das Verzerrung-Varianz-Dilemma

Ein Balanceakt?



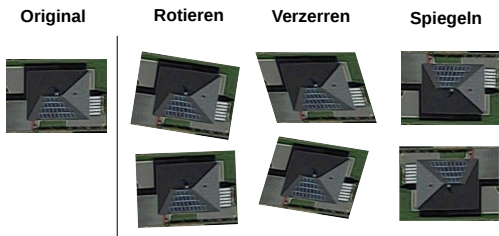
- Pro Epoche **Training und Test**
- Trainingsmenge vergrößern (auch künstlich)
- Modifikation des Trainingsalgorithmus gegen Overfitting: **Weight Decay**
- Nicht alle Parameter (Verbindungen) beim Training beachten: **Dropout Layer**

Overfitting

Strategien dagegen

- Netz verkleinern? Nein: größere Modelle sind **mächtiger**
- Trainingsmenge (künstlich) durch **Variationen** vergrößern

```
ImageDataGenerator(horizontal_flip=True, vertical_flip=True,  
                    rotation_range=10, shear_range=0.2, zoom_range=0.2,  
                    fill_mode='nearest')
```



Overfitting

Strategien dagegen

- Netz verkleinern? Nein: größere Modelle sind **mächtiger**
- Trainingsmenge (künstlich) durch **Variationen** vergrößern
- **Weight Decay/L2 Regularization** Netz lernt kleinere Gewichte, vermeidet Extremwerte

$$C' = C - \frac{\lambda}{2n} \sum_w w^2$$

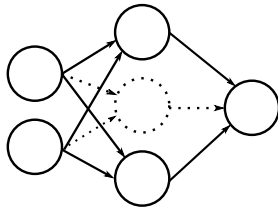
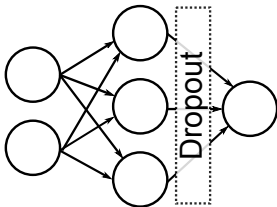
```
from keras import regularizers  
model.add(Dense(42, kernel_regularizer=regularizers.l2(0.01)))
```

- **Dropout Layer**

Overfitting

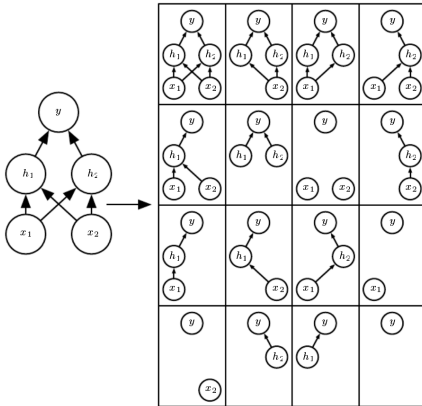
Strategien dagegen

- Netz verkleinern? Nein: größere Modelle sind **mächtiger**
- Trainingsmenge (künstlich) durch **Variationen** vergrößern
- **Weight Decay/L2 Regularization**
- **Dropout Layer**
 - Pro Minibatch: Zufällig 50% der Neuronen entfernen, Training durchführen
 - Vollständiges Netz für reguläre Benutzung
 - Verhindert Co-Adaption, robusteres Lernen bei gleicher Mächtigkeit



Dropout

Bag of Ensemble ANNs



- Verhindert Co-Adaption, robusteres Lernen bei gleicher Mächtigkeit
- Trainiert *Ensemble* von ANNs

Input Layer $p \approx 0.2$

Hidden Layer $p \approx 0.5$

Stiftung Lernalgorithmmentest

Gebräuchliche Algorithmen, und Erweiterungen

Momentum Schluchten für SGD problematisch, gibt „Schwung“

Nesterov Gradient Sensibilisierung des Momentums (große/kleine Schritte)

$\gamma \approx 0.9$; Logarithmisch (0.7–0.9, 0.9–0.95, 0.95–0.97)

Adadelta Adaptiert Lernrate an Parameter
(kerne Lernrate mehr)

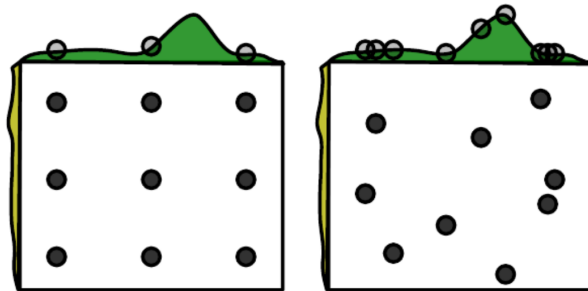
Adam Lernrate + Momentum
 $\beta_1 \approx 0.9$, $\beta_2 \approx 0.999$, $\epsilon \approx 10^{-8}$

Nadam Adam mit Nesterov Momentum
(Parameter wie bei Adam)

- **There's no such thing as free lunch**
- Sparse Data: Adadelta, Nadam; große Datenmengen: SGD
- **Erfahrung**

Grid Search

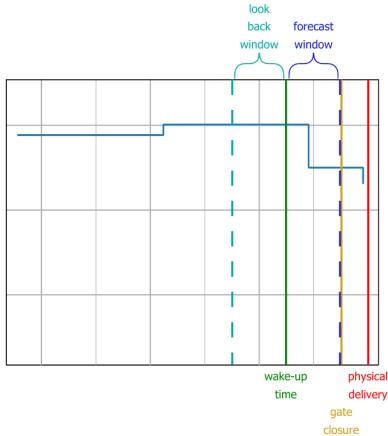
Die Suche nach dem richtigen Hyperparametern



- Grid Search: Suchraster aufbauen
- Dabei sinnvolle Wertebereiche wählen; zumeist **logarithmische Skalen**
- **Hidden Layer:** 50, 100, 200, 500, 1000, 2000
- **Learning Rate:** 0.1, 0.01, 10^{-3} , 10^{-4} , 10^{-5}

Prognose des Intraday-Marktpreises

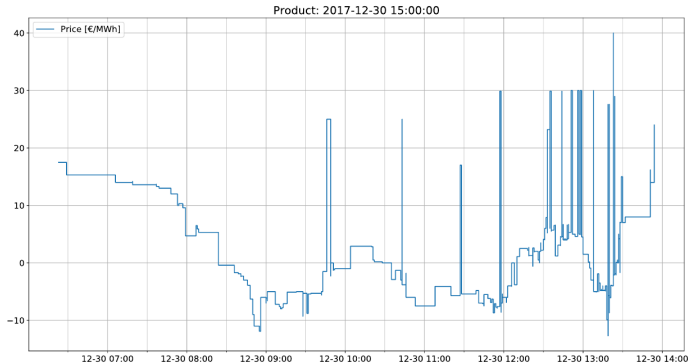
Ein Praxisbeispiel



- **Aufgabe:** Ab *wake-up time* Preis für die nächsten 60 Min prognostizieren
- *forecast-window* aus drei *Bins*: **early**, **mid** und **late**
- **Datenquelle:** *look-back window*

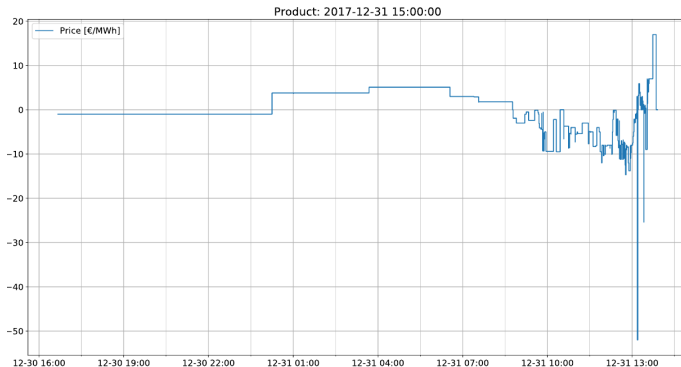
Marktverhalten

Exploration zur Merkmalsselektion



Marktverhalten

Exploration zur Merkmalsselektion



Merkmalsselektion

Eingabedaten der Prognose

Date	Market Area Buy	Market Area Sell	Hour from	Hour to	Volume (MW)	Price (EUR)	Timestamp
				.			
31/12/2017	DE	AT	3	3	1.4	0.10	30/12/2017
31/12/2017	DE	DE	1qh3	1qh3	0.5	-2.00	22:26:00
31/12/2017	DE	DE	15qh1	15qh1	0.7	-1.40	30/12/2017
				.			22:26:00
				.			

Vorverarbeitung

Augmentierung der Eingabedaten

- Umwandlung der Eingabedaten in **äquidistante Zeitreihen**
- Bins haben **Metadaten**

Count Anzahl aggregierter Transaktionen pro Bin

First, Last Wert der ersten/letzten Transaktion

Min, Max Kleinster/größter Wert aller Transaktionen des Bins

Sum Summe aller Transaktionen des Bins

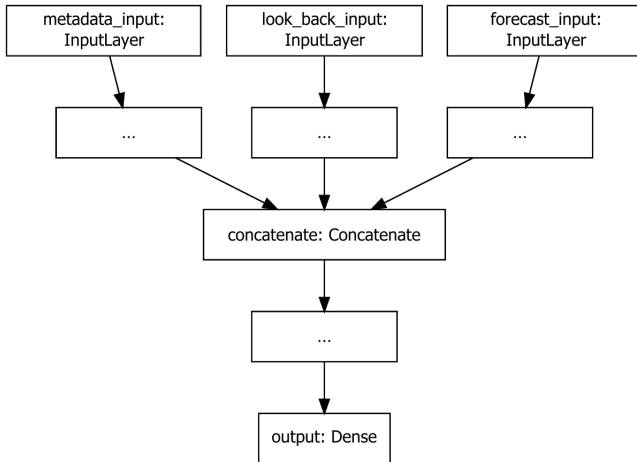
Average Durchschnittswert der Transaktionen

Std Standardabweichung der Werte aller Transaktionen des Bins

Autocorrelation Autokorrelation der Werte aller Transaktionen des Bins

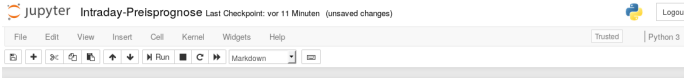
(Korrelation der Zeitreihe mit sich selbst, verschoben um einen Zeitschritt)

Grundsätzliche Struktur der ANNs



Intraday-Preisprognose

Jupyter-Notebook

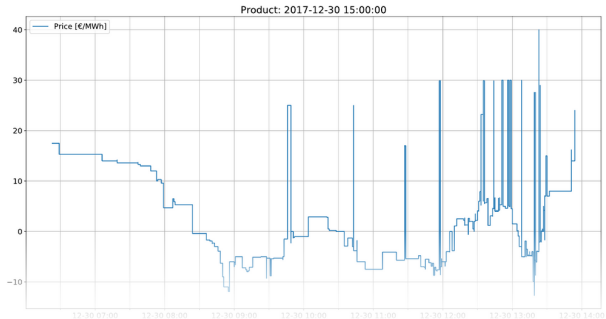


Preisprognose am Intraday-Markt

Einführung

Jeder Prognose liegt die Annahme zugrunde, dass genügend Merkmale – also Eingabedaten – helfen, ein Muster herauszubilden, aus dem zukünftiges Verhalten modelliert werden kann. Grundsätzlich ist eine solche Prognose von einer Vorhersage (wie beispielsweise beim Wetter), bei der durch numerische Berechnungen die Weiterentwicklung aus einem Status Quo simuliert wird, verschieden. Die Deep-Learning-Prognosen approximieren ein Modell des Systems; wenn das Modell umfangreich genug, d.h. also sein Informationsgehalt ausreichend groß ist, stimmen die Ausgaben des Modells, die es aufgrund aktueller Daten macht, mit der Realität überein.

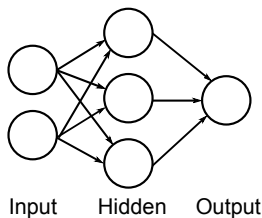
In unserem Fall ist die Realität die Entwicklung des Intraday-Preises am Strommarkt. Die kann im Laufe von zwei aufeinanderfolgenden Tagen stark unterschiedlich aussehen:



Zeitreihenvorhersage mit rekurrenten neuronalen Netzen

Rekurrente Netze

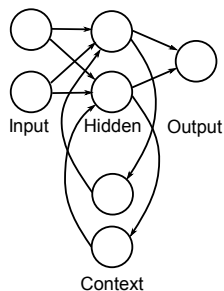
ANN-Architekturen mit Erinnerung



Feedforward Neural Network

Statische Muster

Funktionen



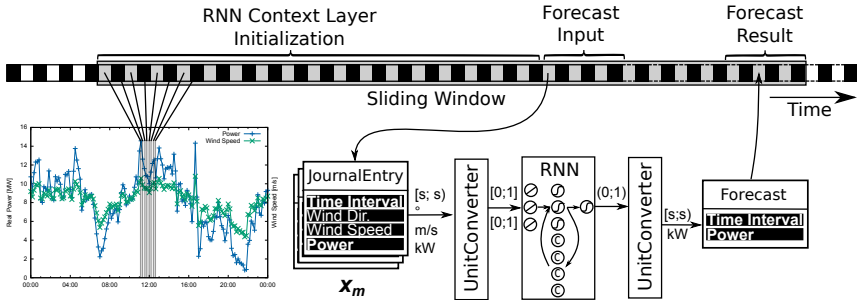
Recurrent Neural Network

Zeitreihen

Dynamische Systeme

Konzept Zeit

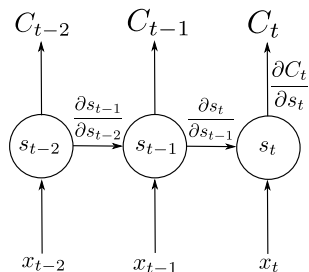
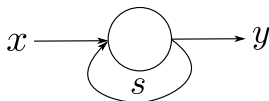
Zeitreihe als relativer Zusammenhang



Backpropagation through Time (BPTT)

Backpropagation für RNNs

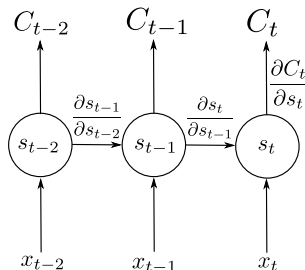
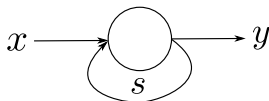
- Normaler Backpropagation-Algorithmus nicht nutzbar (kein DAG)
- Training mit evolutionären Algorithmen
- Alternative: **Auffalten** des Netzes über n Zeitschritte (n : Größe des Zeitfensters)



Backpropagation through Time (BPTT)

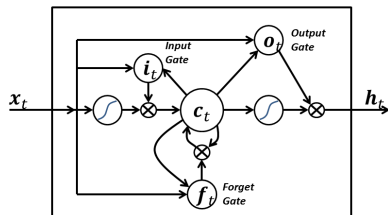
Backpropagation für RNNs

- **Auffalten** des Netzes über n Zeitschritte (n : Größe des Zeitfensters)



- Vorteil: Bekannte gradientenbasierte Verfahren nutzbar
- Nachteil: Wesentlich komplexere Fehlerlandschaft, chaotisches Trainingsverhalten; **verschwindende** oder **explodierende** Gradienten für lange Zeitwischenschritte (*Time Lags*)

Long-Short Term Memory



BiObserver — CC-BY-SA 4.0

- **Gates** beschützen Kontext
- Ein-/Ausgabe: **reguläre, nicht-lineare** Neuronen
- **Forget Gate** kontrolliert Zustand
- **Error Caroussell** am Kontextneuron

$$\mathbf{f}^{(t)} = \sigma_g \left(\mathbf{W}_f * \mathbf{x}^{(t)} + \mathbf{U}_f * \mathbf{h}^{(t-1)} + \mathbf{V}_f \odot \mathbf{c}^{(t-1)} + \mathbf{b}_f \right)$$

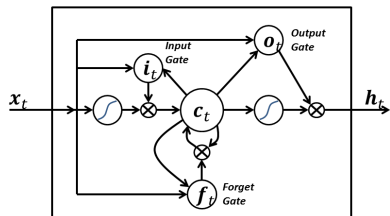
$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \mathbf{c}^{(t)}$$

$$\mathbf{i}^{(t)} = \sigma_g \left(\mathbf{W}_i * \mathbf{x}^{(t-1)} + \mathbf{U}_i * \mathbf{h}^{(t-1)} + \mathbf{V}_i \odot \mathbf{c}^{(t-1)} + \mathbf{b}_i \right)$$

$$\mathbf{o}^{(t)} = \sigma_g \left(\mathbf{W}_o * \mathbf{x}^{(t)} + \mathbf{U}_o * \mathbf{h}^{(t-1)} + \mathbf{V}_o \odot \mathbf{c}^{(t-1)} + \mathbf{b}_o \right)$$

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \sigma \left(\mathbf{W}_c * \mathbf{x}^{(t)} + \mathbf{U}_c * \mathbf{h}^{(t-1)} + \mathbf{b}_c \right)$$

Long-Short Term Memory



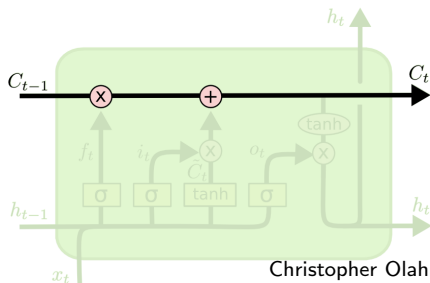
BiObserver — CC-BY-SA 4.0

- **Gates** beschützen Kontext
- Ein-/Ausgabe: **reguläre, nicht-lineare** Neuronen
- **Forget Gate** kontrolliert Zustand
- **Error Caroussell** am Kontextneuron

```
lstm = Sequential()  
lstm.add(LSTM(num_labels,  
              batch_input_shape=(batch_size,  
                                time_steps,  
                                num_features))))
```

LSTM Schritt für Schritt

Kontext-Förderband

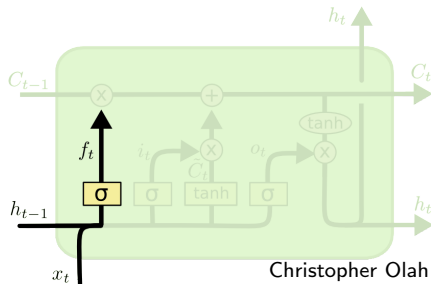


- Status (Kontext) einer Zelle analog einem Förderband

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \sigma \left(\mathbf{W}_c * \mathbf{x}^{(t)} + \mathbf{U}_c * \mathbf{h}^{(t-1)} + \mathbf{b}_c \right)$$

LSTM Schritt für Schritt

Forget Gate

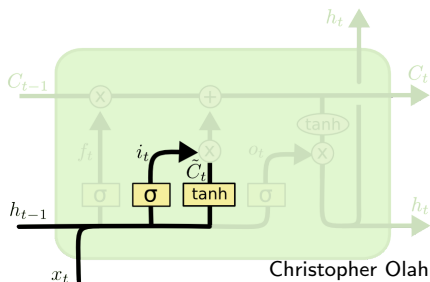


- Zukünftiger Kontext abhängig von Eingabe und voriger Ausgabe
- Differenziertes Behalten/Wegwerfen durch Sigmoid-Funktion
- Erlernte Gewichtsmatrizen steuern Einfluss der Informationen

$$\mathbf{f}^{(t)} = \sigma_g \left(\mathbf{W}_f * \mathbf{x}^{(t)} + \mathbf{U}_f * \mathbf{h}^{(t-1)} + \mathbf{V}_f \odot \mathbf{c}^{(t-1)} + \mathbf{b}_f \right)$$

LSTM Schritt für Schritt

Eingaben

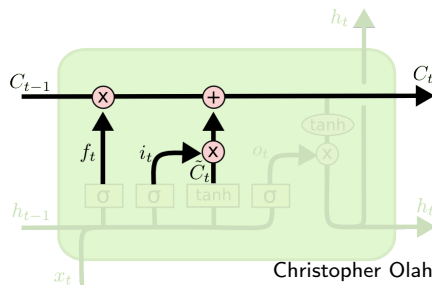


- **Input Gate** bestimmt Relevanz der aktuellen Eingabe
- Addition neuer Informationen zum aktuellen Zustand

$$i^{(t)} = \sigma_g \left(\mathbf{W}_i * \mathbf{x}^{(t-1)} + \mathbf{U}_i * \mathbf{h}^{(t-1)} + \mathbf{V}_i \odot \mathbf{c}^{(t-1)} + \mathbf{b}_i \right)$$
$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + i^{(t)} \odot \sigma \left(\mathbf{W}_c * \mathbf{x}^{(t)} + \mathbf{U}_c * \mathbf{h}^{(t-1)} + \mathbf{b}_c \right)$$

LSTM Schritt für Schritt

Eingaben



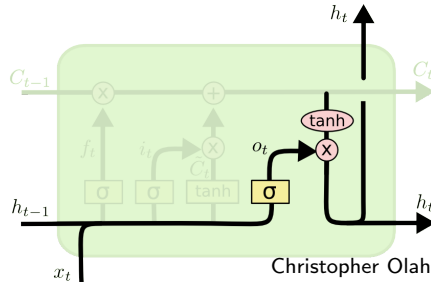
- **Input Gate** bestimmt Relevanz der aktuellen Eingabe
- Addition neuer Informationen zum aktuellen Zustand

$$i^{(t)} = \sigma_g \left(\mathbf{W}_i * \mathbf{x}^{(t-1)} + \mathbf{U}_i * \mathbf{h}^{(t-1)} + \mathbf{V}_i \odot \mathbf{c}^{(t-1)} + \mathbf{b}_i \right)$$

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + i^{(t)} \odot \sigma \left(\mathbf{W}_c * \mathbf{x}^{(t)} + \mathbf{U}_c * \mathbf{h}^{(t-1)} + \mathbf{b}_c \right)$$

LSTM Schritt für Schritt

Ausgaben



- **Input Gate** bestimmt Relevanz der aktuellen Eingabe
- Addition neuer Informationen zum aktuellen Zustand

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \mathbf{c}^{(t)}$$

$$\mathbf{o}^{(t)} = \sigma_g \left(\mathbf{W}_o * \mathbf{x}^{(t)} + \mathbf{U}_o * \mathbf{h}^{(t-1)} + \mathbf{V}_o \odot \mathbf{c}^{(t-1)} + \mathbf{b}_o \right)$$

Multi-Layer Perceptron Einfache Muster, Korrelationen,
Quantifizierungen (z.B. Systemparameter)

CNN (ConvNet) Spaziale, auch temporal-spaziale Muster (Bilder,
Datenreihen mit großem „Fenster“): prägnante Elemente
in Mustern

LSTM Zeitreihenanalyse; weit auseinanderliegende, aber in
Zusammenhang stehende Daten (Time Lag)

GRU Wie LSTM, ggf. schneller im Training/nützlicher für
kleine Datenmengen

Lastgangvorhersage

Ein Praxisbeispiel

jupyter forecasting_residential_load (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

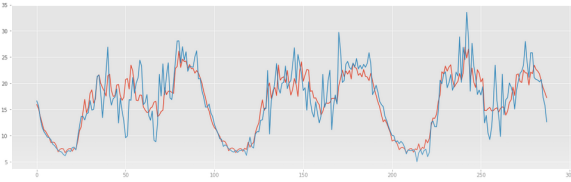
```
5568/5568 [=====] - 7s 1ms/step - loss: 0.0246 - acc: 0.3710
Epoch 9/512
5568/5568 [=====] - 7s 1ms/step - loss: 0.0246 - acc: 0.3763
Epoch 10/512
```

Compute the error of the model on both training and test data set:

```
In [ ]: train_fc = scaler.inverse_transform(lstm.predict(x_train_lstm))
train_exp = scaler.inverse_transform(y_train_lstm)
train_error = np.sqrt(mean_squared_error(train_exp, train_fc)) / np.max(p_res_aggr)
print('Error (nRMSE) of model on training data: %.3f%%' % (train_error * 100))

test_fc = scaler.inverse_transform(lstm.predict(x_test_lstm))
test_exp = scaler.inverse_transform(y_test_lstm)
test_error = np.sqrt(mean_squared_error(test_exp, test_fc)) / np.max(p_res_aggr)
print('Error (nRMSE) of model on test data: %.3f%%' % (test_error * 100))

In [31]: fig, ax = plt.subplots(1, 1, figsize=(20,6))
        _ = ax.plot(test_fc.reshape(test_fc.shape[0] + num_labels, 1)[13 + 96])
        _ = ax.plot(test_exp.reshape(y_test_lstm.shape[0] + num_labels, 1)[13 + 96])
```



```
In [ ]: 
In [ ]:
```

Literaturempfehlungen

Literatur zu Deep Learning

Géron, Aurélien. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Inc., 2017.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

Lapan, Maxim. Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more. Packt Publishing Ltd, 2018.